49th AIAA Aerospace Sciences Meeting, Jan. 4–7, 2011, Orlando, FL

Parallel Adjoint Framework for Aerodynamic Shape Optimization of Component-Based Geometry

Marian Nemec*
Science & Technology Corp., Moffett Field, CA 94035, USA
Michael J. Aftosmis†
NASA Ames Research Center, Moffett Field, CA 94035, USA

We present a parallel adjoint framework for aerodynamic shape optimization problems using an embedded-boundary Cartesian mesh method. The design goals for the framework focus on an efficient and systematic integration of the underlying software modules. By linearizing the geometric constructors used in intersecting triangulated components, we develop a robust approach for computing surface shape sensitivities of complex configurations. The framework uses multilevel parallelism in sensitivity computations. Serial and parallel codes are executed concurrently to speedup gradient computations. A variety of optimizers are supported, and geometric modelers can be either CAD-based or CAD-free. We present design examples involving sonic-boom mitigation and nacelle integration for transport aircraft involving over 160 design variables and using up to 256 processors. This is an important step in our research toward making aerodynamic shape optimization tools available to the broader aerodynamics community instead of CFD specialists only.

I. Introduction

The role of Euler and Navier–Stokes simulations in aerodynamics is broadening from single-point solutions to trend analysis and large parametric studies. Despite this progression, adoption of high-fidelity numerical simulation tools for optimization has been comparatively restrained in most engineering settings. Ultimately, long problem setup and design cycle times make them uncompetitive at providing practical improvements to expert aerodynamicists. Advances in sensitivity analysis via the adjoint method[1–4] have dramatically shortened the design cycle time, but a number of important challenges still remain before high-fidelity aerodynamic shape optimization tools are widely adopted.

Effective use of computational fluid dynamics for optimization requires designers with an in-depth knowledge of a wide and disparate tool-set. In addition to knowledge of the design problem itself, expertise in geometry parameterization, meshing, flow analysis, optimization and high-performance computing are all required. Some of these requirements stem from the lack of automation within the analysis procedure, in particular, surface and volume mesh generation. Unlike database runs where all geometries are known *a priori* and meshes can be preprocessed, optimization runs with many design variables require lengthy setup and careful scripting to ensure automatic execution. As a result, shape optimization is applied only after many analysis studies that allow the user to gain familiarity with the problem and the mechanics of the setup. Often these mechanics are such that the tools can only be narrowly applied before expert intervention is required. This is counter to the fundamental purpose of a design tool.

Integrating the heterogeneous mix of codes involved in optimization into an effective framework is a challenging task. The execution of several highly tuned parallel solvers must be orchestrated with an array

*Senior Research Scientist, Advanced Supercomputing Division, MS 258-5; marian.nemec@nasa.gov. Member AIAA.
†Aerospace Engineer, Advanced Supercomputing Division, MS 258-5; michael.aftosmis@nasa.gov. Associate Fellow AIAA.

Paper 2011-1249

of in-house and commercial serial codes over diverse computational resources. Small missteps in synchronization can have a major impact on performance. Moreover, frameworks that include sensitivity analysis tend to restrict creativity. For example, only a limited set of objective functions and constraints may be preprogrammed and differentiated. There may be essentially no flexibility in choosing a geometric modeler or freedom to select appropriate parameterizations. These choices are precisely the aspects which need to be the most accommodating.

The purpose of the current work is to address many of these challenges by exploring the following ideas:

- Component-based geometry for flexibility in parameterization and in choice of geometric modeler
- Multilevel parallelism for performance
- Framework synthesis with a design description markup for data-flow control
- Lightweight, script-based integration for maintaining modularity of software components
- Symbolic definition of objectives and constraints for general problem specification

We use an adjoint formulation in conjunction with an embedded-boundary Cartesian mesh method that was developed in previous work.[5] A key feature of such methods is the decoupling of the surface triangulation from the volume mesh. This has important consequences in sensitivity analysis. It implies that the shape sensitivity of the triangulation (the change in the location of vertices due to a design variable variation) decouples from the linearization of the volume mesh generator. This greatly simplifies implementation of different geometric modelers and shape parameterization techniques. Previous test cases[5] have demonstrated the use of multiple modelers, including computer-aided design (CAD); however, those examples were limited to single component optimizations, such as an isolated wing. We generalize this approach to component-based geometry.

In our component-based approach,[6] each component in a configuration is a closed surface triangulation. The wetted surface of a full configuration is produced by a Boolean addition of these components. For example, an airplane configuration may be constructed by adding together the surface triangulations of the fuselage, wings and tail components. This approach has several advantages over the traditional approach that requires the geometry modeler to generate a monolithic triangulation constrained by the intersection curves of the components. Different modelers can be used for different components, legacy triangulations (components whose modelers are not available) can be mixed with parametric components, and triangulations of static components (components with no design variables) can be cached. Moreover, triangulation shape sensitivities can be generated independently for each component.

In the area of framework integration, we build on our past experience of constructing a custom optimization framework for both gradient-free and gradient methods (with finite-difference gradient approximations).[7] We found that a script-based implementation works well for enabling multilevel parallelism. We use coarse-grained parallelism to execute many serial tasks in parallel as well as to execute many parallel tasks simultaneously on subsets of the available processors. Moreover, an important aspect of the original framework was its ability to allocate resources on-the-fly. This masked geometry-processing bottlenecks, which may occur if the number of geometry modelers is limited (for example too few CAD licenses) by dynamically determining how many cases to run concurrently on which resources. We use similar ideas in the adjoint framework developed in this paper, where we co-process triangulation shape sensitivities to maintain scalability.

This paper is organized as follows. In the next section we define the optimization problem and briefly review the computation of the objective function and its gradient. We focus on triangulation shape sensitivities and discuss the implementation for component-based geometry. Thereafter, we explain the construction of the parallel adjoint framework that includes multilevel parallelism and a new design description markup. We conclude with three representative examples. The examples demonstrate the capability to use various modelers and the effectiveness of the framework on complex geometry problems with many design variables, and also study the parallel efficiency of the framework. The first two examples demonstrate sonic boom mitigation via an inverse design formulation for a business jet. The final example considers a nacelle-pylon integration problem for a transport airplane.

II. Optimization Problem

The aerodynamic optimization problem we consider consists of determining values of design variables, X[a], that minimize a given objective function

$$\min_X \ \mathcal{J}(X, Q) \tag{1}$$

where $\mathcal{J}$ represents a scalar objective function defined by pressure integrals, for example lift or drag, and $Q = [\rho, \rho u, \rho v, \rho w, \rho E]^{\mathrm{T}}$ denotes the continuous flow variables. The flow variables satisfy the three-dimensional, steady-state Euler equations of a perfect gas within a feasible region of the design space Ω

$$\mathcal{F}(X, Q) = 0 \qquad \forall\ X \in \Omega \tag{2}$$

which implicitly defines $Q = f(X)$. The problem may also involve constraint equations C_j:

$$C_j(X) \leq 0 \qquad j = 1, \ldots, N_{\mathrm{c}} \tag{3}$$

that are independent of the flow state, such as thickness and cross-sectional area constraints[b].

The optimization problem is solved through use of a gradient method. We use a discrete approach, where Eqs. 1- 3 are first discretized and then linearized to obtain the gradient $\mathrm{d}\mathcal{J}/\mathrm{d}X$. In the following sections, we first present a brief outline of the flow solution methodology for evaluating the objective function and then describe the computation of the gradient.

A. Objective Function Evaluation

At each step of the optimization procedure, we compute an approximation of the objective function $\mathcal{J}$ by solving the flow equations, Eq. 2, on a multilevel Cartesian mesh with embedded boundaries. The mesh consists of regular Cartesian hexahedra everywhere, except for a layer of body-intersecting cells, or *cut-cells*, adjacent to the boundaries as illustrated in Fig. 1. The spatial discretization of Eq. 2 uses a cell-centered, second-order accurate finite volume method with a weak imposition of boundary conditions, resulting in a system of equations

$$\mathbf{R}_H(\mathbf{Q}_H) = 0 \tag{4}$$

where H represents the average cell size and $\mathbf{Q} = [\bar{Q}_1, \bar{Q}_2, \ldots, \bar{Q}_N]^{\mathrm{T}}$ is the discrete solution vector of the cell-averaged values for all N cells of the mesh. Steady-state solutions are obtained using a five-stage Runge–Kutta scheme with local time stepping, multigrid and a domain decomposition scheme for parallel computing.[8–10]

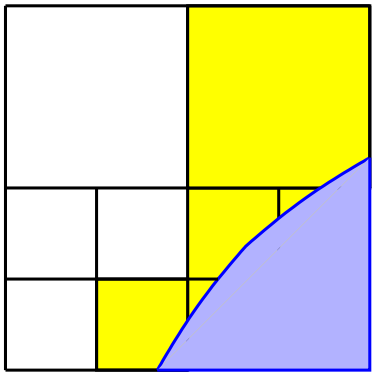

Figure 1. Multilevel Cartesian mesh in two-dimensions with a cut-cell boundary.

B. Gradient Computation

To compute the discrete gradient, $\mathrm{d}\mathcal{J}_H/\mathrm{d}X$, we note that a variation in X influences the computational mesh $\mathbf{M}$ and the flow solution $\mathbf{Q}$. We rewrite Eq. 4 to explicitly include the design variables and the mesh, resulting in a system of equations

$$\mathbf{R}(X, \mathbf{M}, \mathbf{Q}) = 0 \tag{5}$$

where we omit the subscript H to simplify the notation. The influence of shape design variables on the residuals is implicit via the computational mesh

$$\mathbf{M} = f[\mathbf{T}(X)] \tag{6}$$

where $\mathbf{T}$ denotes a triangulation of the wetted surface. The design variables that appear directly in Eq. 5 involve parameters that do not change the computational domain, such as the Mach number and angle

[a] We assume X is a scalar to simplify notation of partial and total derivatives.

[b] Constraints that depend on the flow state are lumped into the objective function via quadratic penalty terms to minimize computational cost by avoiding additional adjoint solutions (see Eq. 8).

of attack. The gradient is obtained by linearizing the objective function, $\mathcal{J}(X, \mathbf{M}, \mathbf{Q})$, and the residual equations, resulting in the following expression

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}X} = \frac{\partial \mathcal{J}}{\partial X} + \underbrace{\frac{\partial \mathcal{J}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{T}} \frac{\partial \mathbf{T}}{\partial X}}_{\mathrm{A}} - \psi^{\mathrm{T}} \left(\frac{\partial \mathbf{R}}{\partial X} + \underbrace{\frac{\partial \mathbf{R}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{T}} \frac{\partial \mathbf{T}}{\partial X}}_{\mathrm{B}} \right) \tag{7}$$

where the vector ψ represents the adjoint variables given by the following linear system

$$\left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}} \right]^{\mathrm{T}} \psi = \frac{\partial J}{\partial \mathbf{Q}}^{\mathrm{T}} \tag{8}$$

The solution algorithm for the adjoint equation leverages the time-marching scheme and parallel multigrid method of the flow solver.[11]

The most interesting part of the gradient computation is the evaluation of terms A and B in Eq. 7. Scanning these triple-product terms from left to right, the linearization with respect to $\mathbf{M}$ involves the flow solver, i.e. Eq 5, while the middle term involves the mesh generator. In embedded-boundary Cartesian mesh methods, an infinitesimal perturbation of the boundary shape affects only the cut-cells. Unlike body-conforming approaches, there is no mesh perturbation scheme to smoothly map boundary shape changes into the interior of the volume mesh. Consequently, the mesh-sensitivity term $\partial \mathbf{M}/\partial \mathbf{T}$, which contains the linearization of the Cartesian-face areas and centroids, volume centroids and the wall normals and areas with respect to the surface triangulation, is non-zero only in cut-cells. This results in a fast and robust procedure for gradient computation; however, there is a reduction in the order of accuracy of the gradient.[5]

The crux in the evaluation of $\partial \mathbf{M}/\partial \mathbf{T}$ is the linearization of the geometric constructors that define the intersection points between the surface triangulation and the Cartesian hexahedra. We explain the steps of the linearization using the example shown in Fig. 2, where a Cartesian hexahedron is split into two cut-cells by the surface triangulation. We require the linearization of the intersection points that lie on Cartesian edges, e.g. point A, and also those that lie on triangle edges, e.g. point B. Focusing on point B, its location along the triangle edge V_0V_1 is given by

$$B = V_0 + s(V_1 - V_0) \tag{9}$$

where s denotes the distance fraction of the face location relative to the vertices V_0 and V_1. The linearization of this geometric constructor is given by

$$\frac{\partial B}{\partial X} = \frac{\partial V_0}{\partial X} + s\left(\frac{\partial V_1}{\partial X} - \frac{\partial V_0}{\partial X}\right) + (V_1 - V_0)\frac{\partial s}{\partial X} \tag{10}$$

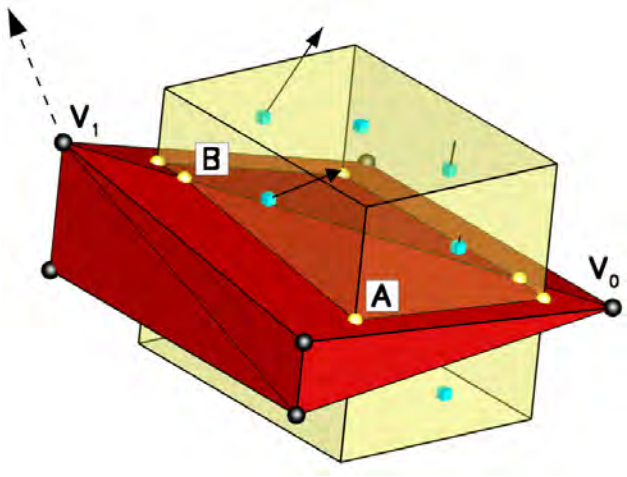

Figure 2. Sensitivity of face centroids (solid vectors) to perturbation of vertex V_1 (dashed vector).

A similar constructor is used for point A.[5] An example linearization is shown in Fig. 2 for the position sensitivity of Cartesian face centroids. An advantage of this formulation is that the dependence of the mesh sensitivities $\partial\mathbf{M}/\partial\mathbf{T}$ on the sensitivities of the surface triangulation $\partial\mathbf{T}/\partial X$, i.e. the third term of the triple products in Eq. 7 and terms $\partial V_i/\partial X$ in Eq. 10, is determined on-the-fly for each instance of the surface geometry. Put another way, there is no requirement for a one-to-one triangle mapping as the surface geometry changes. Consequently, it is relatively easy to accommodate different geometry modelers and allow topology changes and refinement of the surface triangulation between design iterations.

The above formulation isolates the computation of the triangulation shape sensitivities, $\partial\mathbf{T}/\partial X$. This term is tightly coupled to the designer's choice of the geometry modeler and parameterization. We frequently use finite-difference approximations to compute this term. For parametric CAD models, the use of finite-differences is more complicated because the fidelity and connectivity of the triangulation may change for any finite step-size. To circumvent this issue, we use a parametric (u, v) surface-patch mapping.[12] In practice, model regeneration and sensitivity computation are difficult to automate, especially in situations where a configuration involves complex parts with implicit lines of intersection. In the next section, we describe a robust, component-based approach that significantly simplifies the work of geometry modelers by intersecting component triangulations and propagating shape sensitivities to generate a watertight assembly.

III. Component-Based Sensitivities

We explain the computation of component-based sensitivities through use of an optimization example. Consider an airplane configuration that involves the wing sweep as a design variable, as shown in Fig. 3. The configuration may contain many components, such as the tail and engines, but the shape modifications due to sweep affect only the main wing. In a component-based approach, the configuration is assembled from the current instance of its components. Assembly is actually a two step procedure. The first step is a simple union of the components (each with their own set of shape-sensitivities). In the second step, the wetted surface of the entire configuration is extracted via a Boolean summation of the components. The summation includes removal of any internal portions of the geometry where the components overlap. This involves fast and robust surface-surface intersection followed by re-tessellation along the lines of intersection.[6,13] Shape sensitivities on individual components must be propagated through this intersection step to the new vertices added along the lines of intersection.

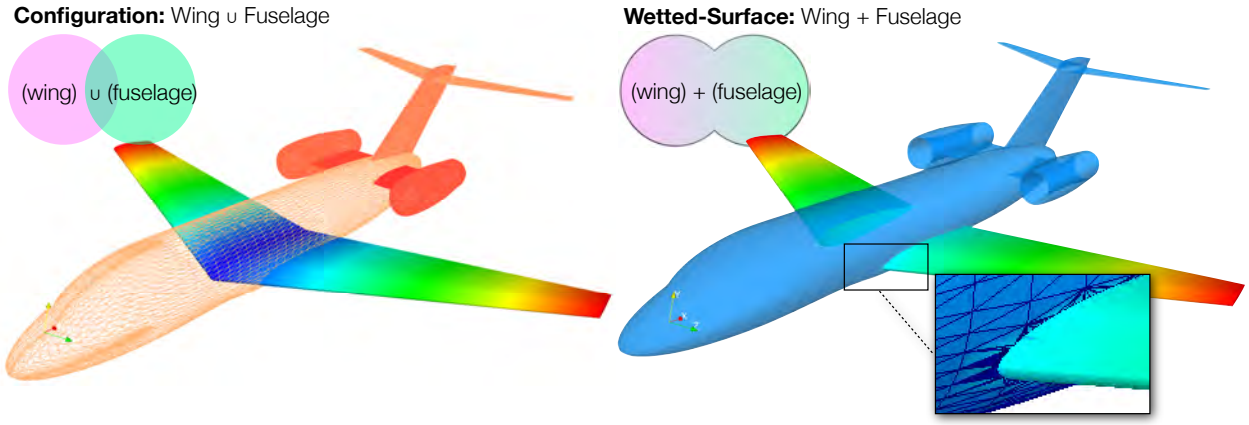

Figure 3. Illustration of a configuration with multiple components and shape sensitivities. The design variable is wing sweep and the contours show sensitivity of the triangulation to sweep variation (streamwise direction). The left side shows the configuration before intersection and the right side is the extracted wetted surface with sweep shape sensitivity. Inset on right shows close-up of the wing-body junction.

Referring to Fig. 3, the color contours on the wing represent the change in the streamwise direction[c] of the wing's vertex positions due to a variation in sweep. We observe that the vertices at the wing root have no sensitivity and stay fixed, while vertices at the tip are most sensitive. The frame at the left shows that prior to intersection the fuselage has no sensitivity to wing sweep, since the sweep of the wing is not associated with the fuselage component. When the component union is formed, the fuselage allocates space

[c] Along the fuselage, nose-to-tail.

for the design variables of all the other components, but these sensitivities are initially all zero. During the extraction of the wetted surface, the triangles that participate in the intersection have non-zero sensitivities to design variables on both components. This is because the location of a vertex shared by both components is sensitive to changes in either component. Consequently, we observe in Fig. 3 that the triangulation sensitivity of the fuselage to sweep is zero everywhere except for the set of triangles that intersect the two components. Similarly, the wing sensitivity to sweep is unchanged everywhere except at the intersecting triangles. In other words, the influence of sensitivities of one component on another is restricted to the triangles participating in the intersection. In addition, a practical detail shown in Fig. 3 is that all static components of a configuration, in this case fuselage, tail and engines, can be pre-intersected into an intermediate configuration to minimize the computational cost of the optimization.

The problem of intersecting two closed, triangulated volumes can be reduced to repeated tests of triangle-triangle intersections.[13] The basic case is illustrated on the left side of Fig. 4 that shows an intersection of two triangles in general position owned by different components. Focusing on the triangle $\mathbf{T}(\mathbf{a},\mathbf{b},\mathbf{c})$[d] and the edge $\overline{\mathbf{de}}$, the right side of Fig. 4 shows the resulting intersection point $\mathbf{P}$. The key step is the linearization of

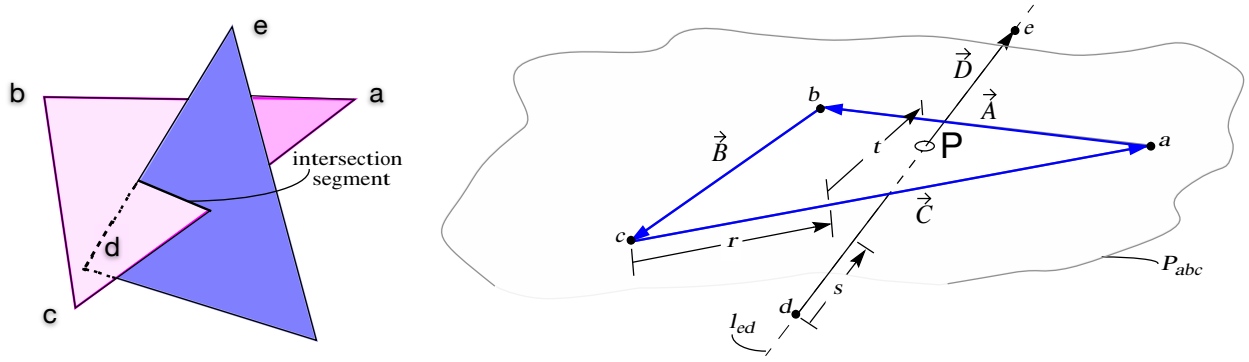

Figure 4. Illustration of a general triangle-triangle intersection between two components (left) and details of an edge-triangle intersection for construction of point P (right). Figures adopted from Ref.[13]

the geometry constructor for point $\mathbf{P}$. This is similar to linearizations we already performed in the volume mesh generator for the intersection of a cut-cell with a triangle[12] (see Eq. 9). We proceed by considering the parametric equations for the edge-triangle intersection. Referring to the right side of Fig. 4, the parametric representation for the plane of triangle $\mathbf{T}(\mathbf{a},\mathbf{b},\mathbf{c})$ using scalar parameters (r,t) is given by

$$\mathbf{T}(r,t) = \mathbf{c} + r\mathbf{C} - t\mathbf{B} \tag{11}$$

where $\mathbf{B} = \mathbf{c} - \mathbf{b}$ and $\mathbf{C} = \mathbf{a} - \mathbf{c}$. Similarly, the parametric representation of the line segment $\overline{\mathbf{de}}$ is given by

$$\mathbf{l}(s) = \mathbf{d} + s\mathbf{D} \tag{12}$$

where $\mathbf{D} = \mathbf{e} - \mathbf{d}$. The pierce point location $\mathbf{P}(s^\star)$ is defined by the intersection of the line and plane

$$\mathbf{P} = \mathbf{d} + s^\star\ \mathbf{D} \tag{13}$$

with $s^\star = f(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e})$. Solving the parametric equations for the point of intersection yields a value of s^* given by

$$\begin{aligned} s^\star = \frac{1}{\Gamma}\{&(c_0-d_0)[(a_2-c_2)(c_1-b_1)-(a_1-c_1)(c_2-b_2)] \\ &-(c_1-d_1)[(a_2-c_2)(c_0-b_0)-(a_0-c_0)(c_2-b_2)] \\ &+(c_2-d_2)[(a_1-c_1)(c_0-b_0)-(a_0-c_0)(c_1-b_1)]\} \end{aligned} \tag{14}$$

where

$$\begin{aligned} \Gamma = \{&(e_0-d_0)[(a_2-c_2)(c_1-b_1)-(a_1-c_1)(c_2-b_2)] \\ &-(e_1-d_1)[(a_2-c_2)(c_0-b_0)-(a_0-c_0)(c_2-b_2)] \\ &+(e_2-d_2)[(a_1-c_1)(c_0-b_0)-(a_0-c_0)(c_1-b_1)]\} \end{aligned} \tag{15}$$

[d]Bold type denotes Cartesian vectors.

The linearization of the constructor, Eq. 13, for the intersection point $\mathbf{P}$ is given by

$$\frac{\partial \mathbf{P}}{\partial X} = \frac{\partial \mathbf{d}}{\partial X} + s^\star \frac{\partial \mathbf{D}}{\partial X} + \frac{\partial s^\star}{\partial X}\mathbf{D} \tag{16}$$

which represents the change in the location of $\mathbf{P}$ along the edge $\overline{\mathbf{de}}$ as a function of shape sensitivities at the vertices $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ of the first component and vertices $(\mathbf{d}, \mathbf{e})$ of the second component with respect to design variable X. The linearization is straightforward but tedious due to Eqs. 14 and 15. Note that this is a linearization of a discrete operator that projects the component-based sensitivities onto the new vertices of the resulting triangulation. Consequently, as shown in the inset of Fig. 3, this may impact the smoothness of the sensitivities on the intersection line, thereby reducing the order of convergence of the gradient for some design variables.[5] Once all intersection points are computed and linearized, we annotate the triangulation of the wetted surface with its shape sensitivities and proceed to obtain the gradient.

At this stage, we have completely described the evaluation of both the objective function and gradient. We turn our attention to the construction of the optimization framework.

IV. Optimization Framework

A. Performance Considerations

The call sequence of software modules in a design cycle of our gradient-based approach (Sec. II) is straightforward. It is a familiar loop that involves geometry generation, mesh generation, flow solution, objective function evaluation, adjoint solution, gradient evaluation and the optimizer. To maintain parallel efficiency on many-core supercomputers, however, *all* components of the framework require perfect scalability as the problem size increases in the number of design variables, and the sizes of the surface and volume discretizations. This is difficult to achieve both in theory and practice.[14] In our situation, the geometry modeling, mesh generation, and some pre- and post-processing modules are serial codes, while the flow and adjoint solvers are efficient parallel codes.[10,11] In addition, some modules are limited to specific platforms. Multilevel parallelism is a promising strategy for integrating such a heterogeneous mix of codes. The basic idea is to execute serial codes concurrently whenever possible and to execute multiple instances of parallel codes simultaneously on subsets of the available processors to maximize parallel efficiency. This strategy also minimizes modifications to the existing code-base, maintains modularity, scales well from desktops to supercomputers and extends to other optimization approaches.

To help devise an efficient multilevel parallel strategy, we examine the computational work in a typical optimization. The number of design cycles required for a gradient-based method to reach an optimal design is proportional to the the number of design variables, $\mathcal{O}(N_{\mathrm{DV}})$. Within each design cycle, solution of the adjoint equation, Eq. 8, eliminates the need to perform the N_{DV} flow solves that would be required by a finite-difference approach. All the remaining work is associated with forming the partial derivative terms in Eq. 7. In particular, we require N_{DV} computations of surface shape sensitivities, $\partial\mathbf{T}/\partial X$, which means that in a typical optimization the surface shape sensitivities are computed $\mathcal{O}(N_{\mathrm{DV}}^2)$ times. Each computation may involve several model regenerations and triangulations where computational complexity may be worse than linear with respect to the number of triangles in the configuration. We also require $\mathcal{O}(N_{\mathrm{DV}}^2)$ linearizations of the cut-cells for the term $\partial\mathbf{M}/\partial\mathbf{T}$ whose complexity is roughly proportional to the number of triangles and $\mathcal{O}(N_{\mathrm{DV}}^2)$ linearizations of the residual for the term $\partial\mathbf{R}/\partial\mathbf{M}$ with complexity proportional to the volume mesh size.

The key to an effective framework architecture is to exploit the fact that the computation of the partial derivatives is independent for each design variable. Hence, it is possible to parallelize $\mathcal{O}(N_{\mathrm{DV}})$ operations in each design cycle by dynamically reallocating the processors used by the parallel flow and adjoint solvers[e]. Moreover, the computation of the surface sensitivities is independent of the flow state. In other words, shape sensitivity computations can be overlapped with flow and adjoint solutions.

The basic architecture of our adjoint optimization framework exploits all opportunities for the concurrent execution of serial and parallel codes. This is shown in Fig. 5, where we illustrate forks of the main processing thread. The optimizer is treated as a "black-box" and is allocated its own thread, shown on the right-side of Fig. 5, to facilitate usage of different optimizers. The left-side of the figure shows the main components of the analysis and sensitivity modules. Tasks associated with geometry generation and linearization run

[e]We assume that memory is available to hold multiple flow and adjoint solutions.

on a separate thread from the flow and adjoint solvers. Co-processing of shape sensitivities is especially advantageous if the computing resources used by the geometry modelers are independent from the resources dedicated to the flow and adjoint solvers. An example is a client-server CAD interface that we used in previous work.[7] If the geometry modelers share resources with the flow and adjoint solvers, then we complete the geometry generation and linearization step before the flow solver executes, as indicated by the dashed line in Fig. 5. The remaining partial derivative terms in Eq. 7 are evaluated after the adjoint solution. For the results presented in the next section, which use a fixed mesh, the linearization of the cut-cells could also be co-processed with the flow solution; however, if mesh adaptation is used then this separation is not possible.

Figure 6 shows a detailed view of the geometry and analysis modules from Fig. 5. Fine-grained parallelism is used in the execution of the flow and adjoint solvers. For geometry generation and shape sensitivity computations, we use a master-slave scheduler to execute up to N_{DV} geometry modelers concurrently. We also execute the geometry intersection module on a separate thread so that we exploit additional coarse-grained parallelism as sensitivity computations complete. A similar strategy is used in the gradient module (shown in Fig. 5), but since the module associated with the residual linearization is parallel, multiple instances of this module are executed simultaneously on groups of processors.

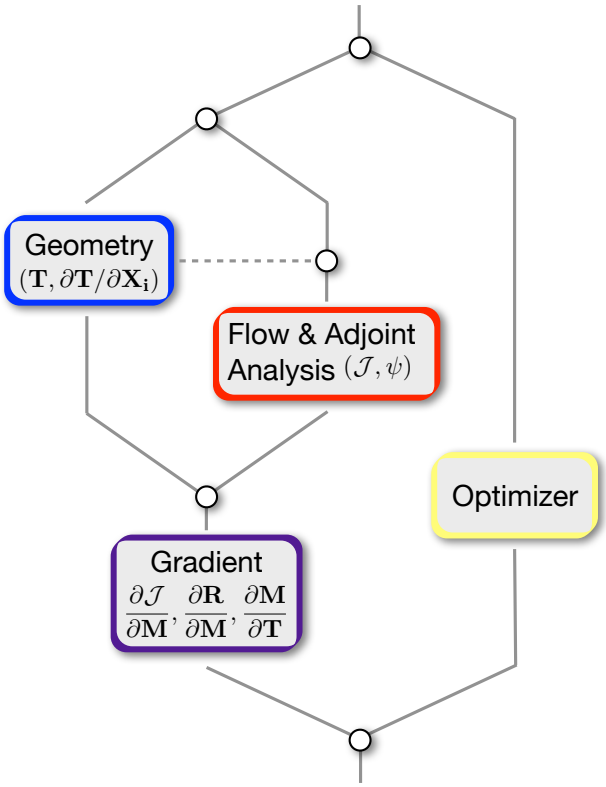

Figure 5. Main processing flow of the adjoint optimization framework.

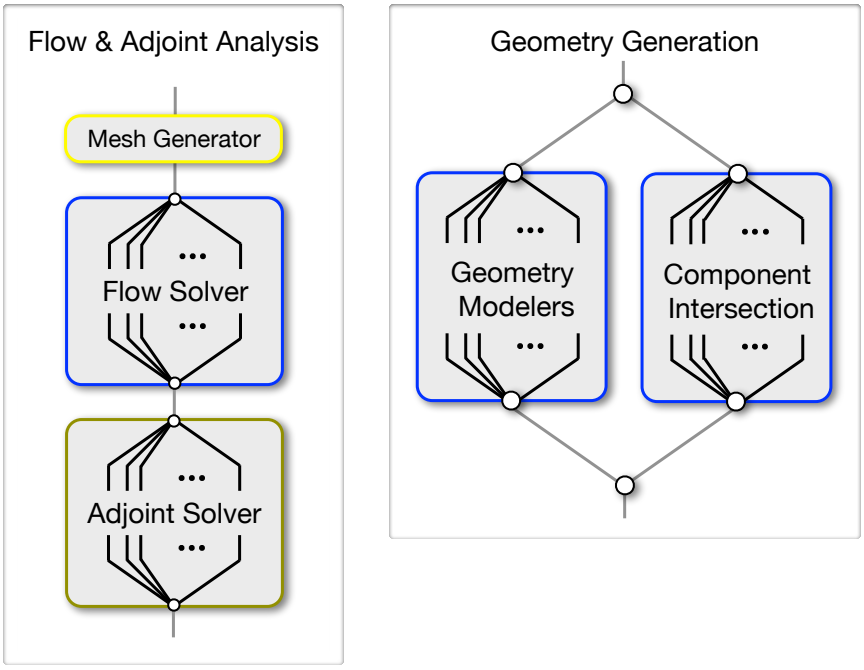

Figure 6. Thread structure of the flow and adjoint analysis module and geometry processing module. Fine-grained parallelism is used by the flow and adjoint solvers (left), while concurrent execution of serial codes is used for geometry modeling and component assembly (right).

B. Extensible Design Description Markup (XDDM)

As the optimization framework grows to involve more codes and handle more general problems, the control of data flow within the framework and the parsing and processing of data by downstream modules can quickly become an entanglement of scripts, files and file formats. To mitigate these issues, we use the Extensible Markup Language (XML) as the primary meta-data format throughout the framework. Previous work[15,16] on the development of XML standards (or schemas) for multidisciplinary optimization focused on web-based applications. The requirements for our framework are much simpler in this regard. Consequently, we define and implement a custom XML syntax referred to as "Extensible Design Description Markup" (XDDM).

Our goal is a terse vocabulary to define design variables, objective functions and constraints. We use these elements in the various software modules to control data flow. This includes specifying the shape of the geometry, defining operating (freestream) conditions, and symbolically specifying and evaluating objectives and constraints in conjunction with their sensitivities. We also introduce elements that are specific to the various software modules within the framework, such as syntax specific to geometry modelers. Taken together, the basic and module-specific elements provide a design database at each iteration of the optimization. A module may query the database file for whatever information it needs and fill-in the appropriate elements and attributes for processing by downstream modules. For example, at the end of a design iteration a gradient-based optimizer reads the values and gradients of the objective function and constraints, and fills-in design variable values for the next design iteration. We provide a brief overview of XDDM in Appendix A, which also contains an example taken from the final test case of this paper.

V. Results

We test the performance of the framework on three design examples. The first two problems involve off-body inverse design at supersonic conditions, while the final example involves drag minimization at fixed lift for a transport aircraft. These problems are chosen since they demonstrate the effectiveness of the framework over a large range of design variables (18-161 variables), a range of mesh sizes (2-17M cells), CAD-based and CAD-free geometry modelers, and constrained and multipoint design problems.

A. Inverse Design for an Attainable Target

Numerical experiments with the design framework begin with examination of an inverse-design problem with a known solution. This first investigation verifies the ability of the design framework to recover a unique known result. We model a sonic-boom optimization problem in which the aircraft shape is driven by prescribing a desired pressure signal in the flow-field some distance away from the body. Figure 7 outlines the basic approach. We prescribe an achievable target pressure signal along a sensor located several characteristic lengths under the vehicle, and use this signal to recover the values of the shape design variables that generated the signal. This formulation has an interesting subtlety. Unlike earlier verification studies with this flow and adjoint solver[5,17] it is the near-field pressure signal that drives the design. Thus the functional is not defined on the body itself and since the off-body mesh is held constant, the mesh sensitivities in Eq. 7 are zero along the off-body sensor.

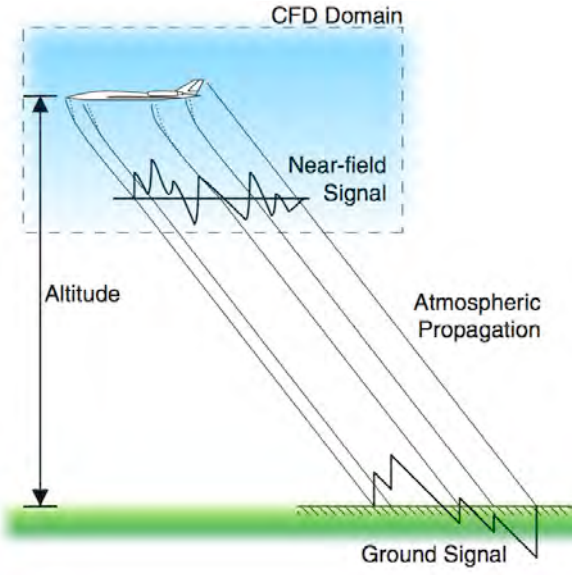

Figure 7. Illustration of the approach for sonic-boom prediction.

Figure 8 shows a generic supersonic business jet model that was generated using the Pro/ENGINEER CAD system. The model is parametric and has approximately 100 independently adjustable parameters to describe the positions and shapes of the fuselage, main wing and empennage components. Since they are generated using a constructive solid-modeling approach, each of the four components is a watertight solid. Figure 9 shows the baseline pressure signature (undertrack signal) of this model at a distance of two-body lengths below the vehicle at a free-stream Mach number $M_\infty = 2.0$ and angle of attack of $0°$.

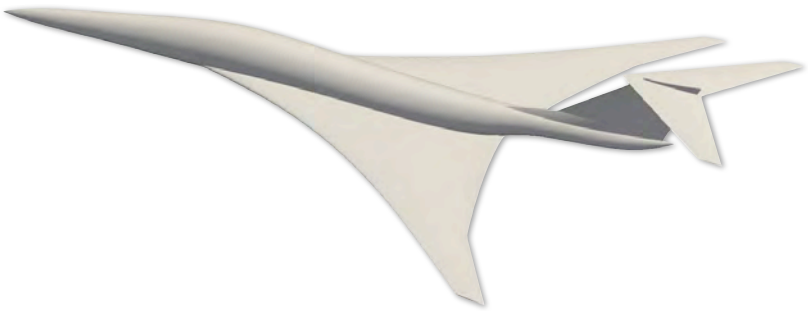

Figure 8. Supersonic business jet baseline geometry. The Pro/ENGINEER model contains four watertight solids and over 100 controllable parameters

The frame on the right of Fig. 9 shows the target pressure signal which provides a goal for the shape optimization. This is an *attainable* target, meaning that it was generated using known values of the design variables, running a simulation and then capturing the signal. Thus we ensure that, properly motivated by the optimizer, the baseline configuration can be driven to this signal. Comparing the baseline and target signatures, the initial signature shows non-smooth forebody compressions and a large pressure rise due to the wing shock. In the optimization target, we seek a smoother forebody compression, a decrease in the amplitude of the main wing-shock and tailoring of the aft-signal to reduce the peaks.

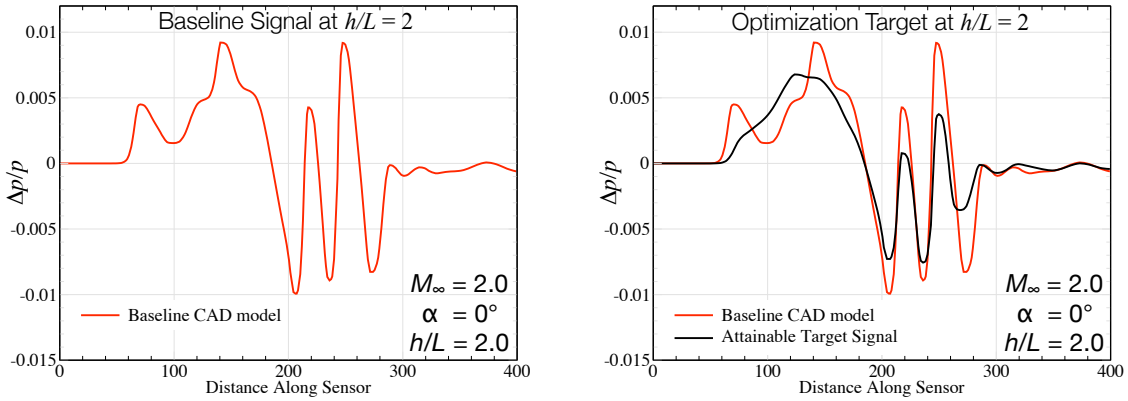

Figure 9. Pressure signatures on the centerline a distance $h/L = 2$ below the model. Initial signature is shown in red and an attainable target pressure signature is shown in black. Sensor location is on the centerline.

1. Design Variables and Objective Function

Of the 100 or so parameters defining the vehicle's shape, 18 were modified to produce the target signal. These same parameters were used as variables in the design process. Figure 10 outlines the major shape parameters governing the vehicle's construction. The fuselage is constructed via an array of circular cross sections. Nine of the design variables are dedicated to controlling the radii and location of these sections. In addition, the incidence of the horizontal tail is allowed to adjust. The wing is lofted spanwise using three airfoil sections between splined leading and trailing edges. Eight of the design variables are dedicated to controlling the wing airfoil shape, with four parameters contouring the lower surface of the wing at both the root and mid-span stations. The wing planform and upper surface are held fixed.

We use the Computational Analysis and Programming Interface (CAPRI) developed by Haimes et al.[18] for control and regeneration of the native CAD model. As these new instances of the geometry are regenerated, surface triangulations are automatically produced for the individual CAD solids[19] and the updated configuration produced via boolean addition of the tessellated solids.[6] Shape sensitivities of the component tessellations $\partial \mathbf{T}/\partial X$ are obtained via finite-differencing of the component surface triangulations by mapping CAPRI's parametric (u, v) surface representation of the locally perturbed shapes to the base state.[5,12] The sensitivities are combined to form the configuration sensitivities using a linearization of the geometric constructors as described earlier.

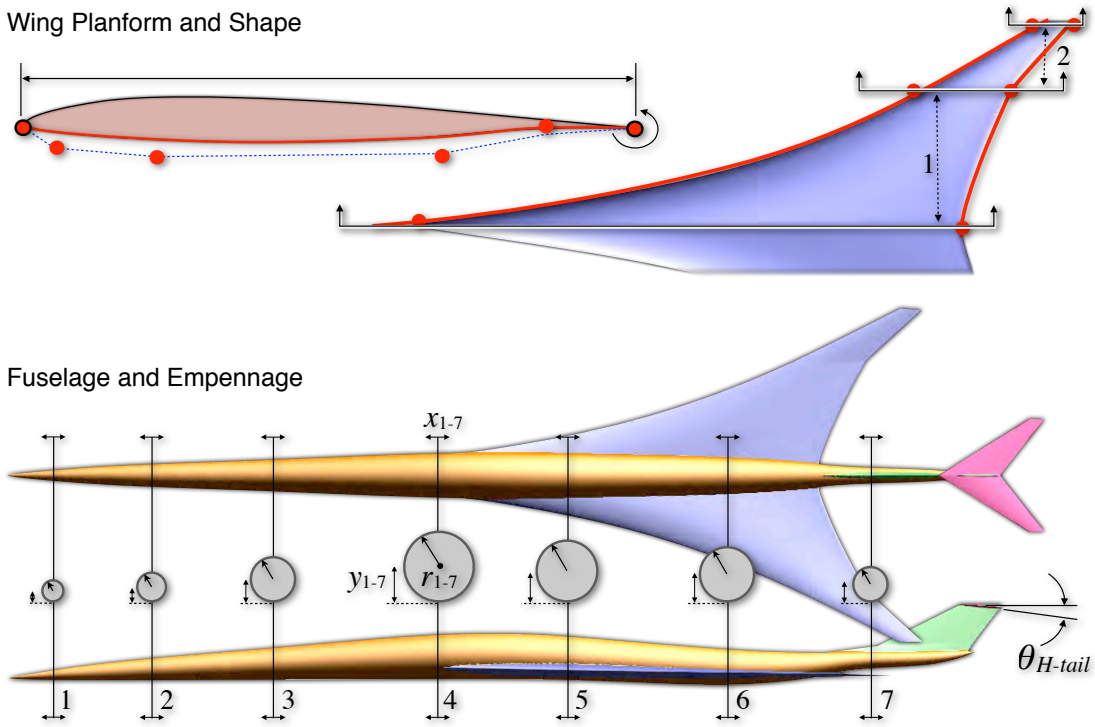

Figure 10. Definition of design variables for sonic-boom optimization examples.

Our objective is to match the desired undertrack pressure signature shown in Fig. 9. To do this, we specify an objective function that seeks to minimize the square of the 2-norm of the difference between the actual pressure signature and the target signature over the sensor. Specifically, we seek to minimize

$$\mathcal{J} = \int (P - P_{\text{target}})^2 dS \tag{17}$$

2. *Optimization Results*

This example was run using the SNOPT (v7.2) optimizer.[20] Optimization continued for a total of 24 major iterations (design cycles) and required approximately 1015 seconds of wall-clock time per design iteration. The magnitude of the objective function was successfully reduced by over four orders of magnitude with no apparent slow-down in convergence. This case was run on 32 dual-core Intel Itanium2 processors (1.6GHz) and meshes for the flow and adjoint solvers had approximately 2.3M cells. Geometry was served using an array of 5 CAPRI CAD servers running Pro/ENGINEER to fulfill the framework's requests for new instances of the design and shape sensitivities for each design variable. The framework's performance is examined in more detail below.

Figure 11 summarizes the results of the optimization. The frame at the left recounts the baseline and target pressure signatures while that in the middle shows the signature after optimization. Agreement is very good and symbols are used on the target curve to make it possible to distinguish the final design from the target signature. Convergence of the objective function is shown in the right frame of Fig. 11. Considering that this example involves non-smooth supersonic flow, geometry with sharp edges, discretely generated shape sensitivities and other real-world concessions which can hamper convergence, the results demonstrated are quite encouraging and verify the framework's ability to reconstitute a known design from a prescribed signature.

Figure 12 shows some details of the geometry after optimization. Frame (a) shows a side-view of the nose and empennage. To smooth the forebody compression, the optimizer has tailored the nose and canopy region. The maximum fuselage diameter has shifted forward to the leading-edge wing-body juncture resulting in a more area-ruled fuselage. The aft-fuselage is more smoothly contoured than the baseline design, and

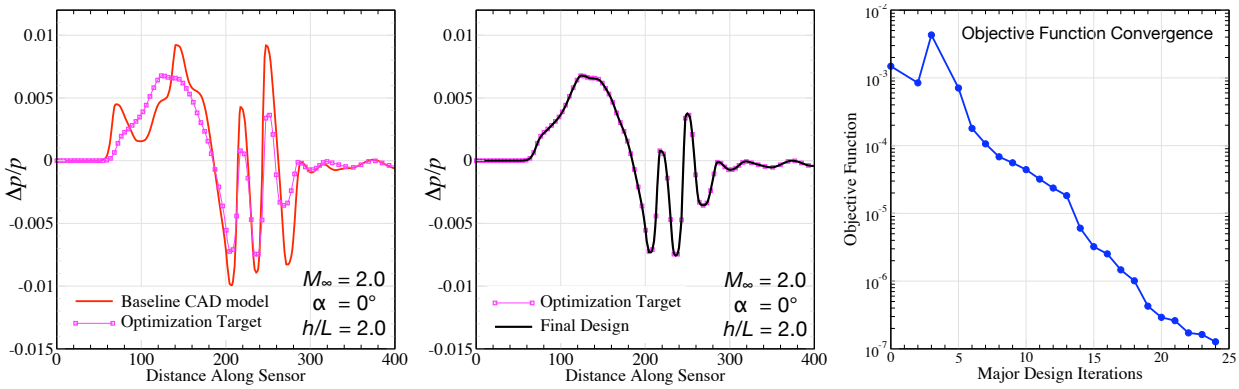

Figure 11. Principal results for attainable optimization example. Left: Baseline and Target pressure signatures at $h/L = 2.0$. Middle: Signature of final design compared with the optimization target. Right: Convergence of the objective function Eq. 17 as the design progressed.

the horizontal tail has been aligned with the local flow. Frame (b) compares the root airfoil section of the optimized and baseline geometries. Thickness has been removed from the leading edge reducing the strength of the wing-shock. Frame (c) shows a rear three-quarter view of the underside of the vehicle. We note substantial re-contouring of the aft fuselage both ahead and aft of the wing trailing-edge. Note also the topology change at the aft wing-body juncture. In the baseline geometry the wing trailing-edge was unbroken by the fuselage. The optimized geometry has smoothed the lower aft fuselage so that it now extends beyond the root wing trailing-edge. In comparing the shape changes with the desired modification to the signal in Fig. 11, we see a more gradual forebody compression and the wing leading-edge overpressure has been weakened by approximately 50%, as have been the peaks in the aft portion of the signature.

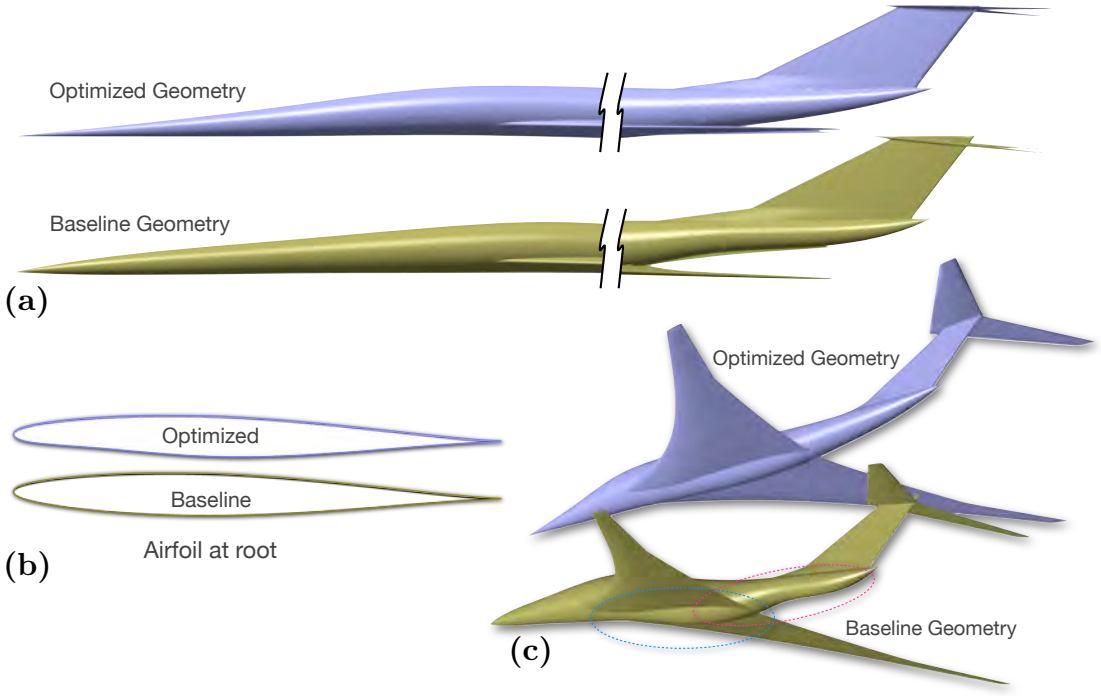

Figure 12. Initial and optimized shapes

3. Framework Performance

This example evolved the design through 24 design cycles and required about 17 minutes of wall-clock per cycle. Each design iteration produced new values of the CAD parameters governing the vehicle's shape, demanding that the CAD system instantiate a new revision of the geometry. In addition to this, the method required shape sensitivities $\partial \mathbf{T}/\partial X$ for every design variable. Since these are discretely generated, providing them demands two more instantiations of the CAD model per design variable. Thus with 18 design variables, the framework requires 37 model regenerations per design iteration. In this case, these were provided through a web services protocol by an array of 5 CAPRI servers running the CAD system. Notice that while a total of 37 regenerations are needed, the flow and adjoint solvers only require the baseline geometry to get started. Thus, with remote geometry farms such as this, the framework overlaps the geometry production needed to feed the shape sensitivities with the parallel computation of the flow and adjoint solves on the main compute engine. This setup effectively masks the cost of generating the component-based shape sensitivities behind the work of the big compute tasks.

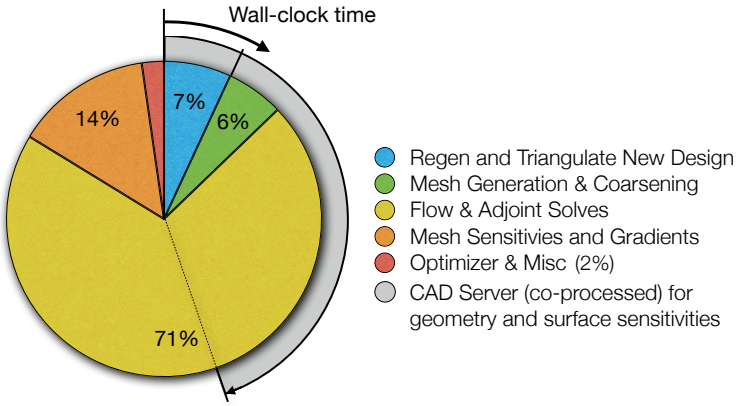

Figure 13. Timing budget for typical optimization cycle of the inverse design example. One design cycle required approximately 1015 wall-clock seconds using 32 dual-core processors.

Figure 13 traces the execution of a typical design cycle to give insight into the framework's performance. The figure traces the cycle clockwise, starting at top of the circle. In the foreground is a pie-chart of the main compute thread (multicolored) while grey shading of the circle in the background shows activity of CAD servers. At the start of each design cycle, requests are passed from the optimizer to the CAD servers for baseline geometry and surface sensitivities. The baseline geometry gets built first and the main compute engine must wait for this geometry before it can proceed. In this case, the servers needed 70 seconds to return the tessellation through CAPRI to the compute engine. Generating a Cartesian mesh for the new surface triangulation requires about 60 seconds, and the mesher runs on a single thread.[6] After meshing is complete, the framework runs both the flow and adjoint solvers. This parallel computation gets distributed over all available threads of the compute engine. In this case the flow solver took about 420 seconds while the adjoint required about 300 seconds. These codes scale very well, and the parallel efficiency when startup and shutdown are included is around 90%. On average the CAD servers required a total of 450 seconds to build all geometry and compute all the component sensitivities. As indicated in Fig. 13 this was successfully masked by the parallel flow and adjoint solves on the main compute engine. After the flow and adjoint solves were complete, the framework computes mesh sensitivities and overall gradients for each of the design variables. These were processed with eight threads each (effectively processing 8 DVs at a time). Parallel efficiency of this step is estimated at around 50% and required an average of 142 seconds. Processing by SNOPT and general housekeeping of logs and directory structure required 13 seconds or about 2% of the design cycle. The fact that the best scaling codes account for the majority of the execution time is encouraging and the net parallel efficiency on the total pool of 64 cores for the entire design cycle was reported to be 65% by the batch queue monitor.

B. Optimization for Boom Signature at Fixed Lift

The second example builds on the use of the off-body functional and considers a realistic inverse-design problem aimed at sonic-boom control. We again drive the aircraft shape by prescribing a desired signature away from the body, but this time we simultaneously impose a lift requiement. While the previous example specified an attainable target signature, this case purposely attempts an unattainable target – a near-field target of zero under-track pressure disturbance.

1. *Design Variables and Objective Function*

We use the same initial model and free stream Mach number as in the previous example ($M_\infty = 2.0$), but this time at 1° angle-of-attack giving a lift coefficient for the baseline model $C_L = 0.094$. In this example, a total of 22 CAD parameters were open for design. The 9 design variables on the wing and horizontal tail were the same as in the previous example, but four additional parameters were included on the fuselage to permit control of the vertical displacement of the fuselage sections. Figure 14 shows the baseline pressure signature (undertrack signal) at a distance of two-body lengths below the vehicle. The increase in angle-of-attack is evident in the baseline signature. Comparing with the 0° profile in Fig. 9 the signal is stronger in both peak values and in the magnitude of the mean disturbance. The zero-disturbance target distribution is shown in the right frame of the figure by the blue dashed line. Since it is such a radical change in the signal, and is not derived from a known shape, this target distribution is obviously unlikely to be attainable. Nonetheless, the problem is interesting since it asks the optimizer to generate the weakest disturbance possible while attempting to maintain the baseline lift of $C_L = 0.094$.

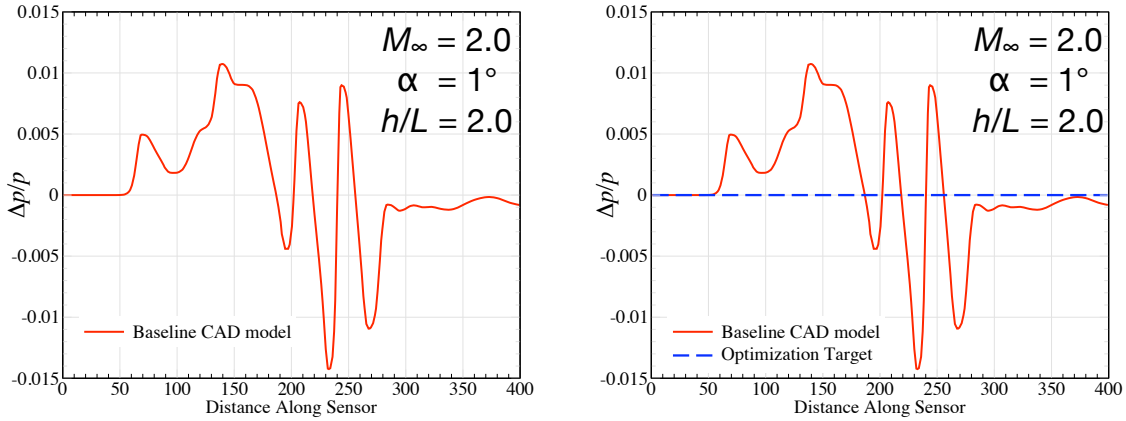

Figure 14. Initial and target pressure signatures for optimization at fixed lift at $M_\infty = 2$ and $\alpha = 1°$. Sensor located at centerline, two body-legths below the vehicle. The target distribution for this example was zero pressure disturbance while maintaining the same lift as the initial vehicle case.

As in the case with the attainable target, the objective function was specified by minimizing the 2-norm of the difference between the actual pressure signature and the target signature over the sensor, and the lift constraint was included through the addition of a penalty term which attacks designs with lift coefficients different from the desired target.

$$\mathcal{J} = \int (P - P_{\text{target}})^2 dS + 8(C_L - 0.1)^2 \tag{18}$$

2. *Optimization Results*

As before, the SNOPT optimizer was used.[20] This example was run for a total of 30 major iterations before convergence of the objective function flattened. This case was run on 64 dual-core Intel Itanium2 processors (1.6GHz) using meshes with approximately 2.3M cells. Geometry was served via web-services from an array of CAPRI CAD servers running Pro/ENGINEER backends. At each design iteration, obtaining surface sensitivities of the geometry required 45 regenerations of the CAD model. Since the framework co-processes the surface sensitivities with the parallel objective function and gradient calculations, the cost of the CAD work was effectively masked by 720 seconds of wall-clock time required for the flow and adjoint solvers. With

6 CAD servers preparing and serving geometry and the main compute hardware responsible for computing objective functions, gradients and meshing and mesh sensitivities, the overall parallel efficiency was measured at 57% by the job scheduler.

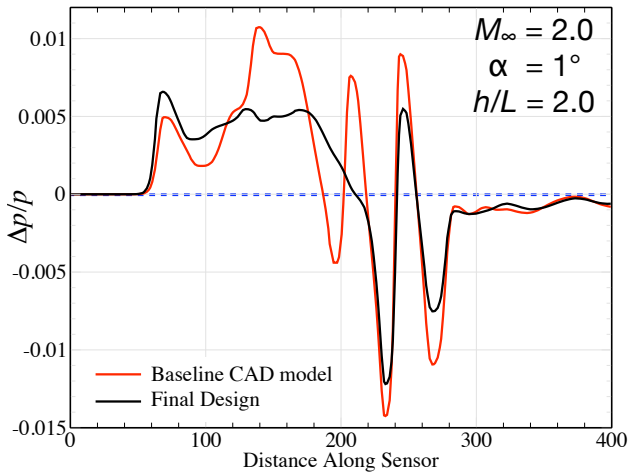

Figure 15. Initial and final near-field signature for optimization at fixed lift at $M_\infty = 2$ and $\alpha = 1°$. Pressure signature measured along the centerline, two body-legths below the vehicle. Target indicated by dashed blue line.

Figure 15 displays the pressure signature along the near-field undertrack sensor ($h/L = 2$) that drove the design before and after optimization. The "target" of zero-pressure disturbance is indicated by a thin dotted line. In examining the final signature, we see that the optimizer has focused most of its efforts on reducing the maxima and minima of the initial signal. Peak over and under pressures were reduced from +0.11 and -0.14 to +0.07 and -0.13 respectively. Moreover, since the functional in Eq. 15 is attacking the square of pressure difference, it has reduced the amount of real estate away from $\Delta p = 0$ substantially flattening the overall signature.

Of course, the functional in Eq.18 was intended to act as a fun and quick surrogate for a true boom functional. To estimate boom, near-field signatures such as those in Fig. 15 are first propagated to the ground using propagation methods.[23–25] Figure 16 shows an estimate of the ground-signature of both vehicles shown as a plot of overpressure vs. time. These ground signals were generated by propagating the near-field signatures (Fig. 15) taken at $h/L = 2$ to a ground plane using the NFBoom propagation code[22,24] using an assumed altitude of 55,000 ft and a vehicle length of 170 ft. The resulting ground-signal is then integrated and transformed to come up with an estimate of boom loudness measured either in PLdB or dBA.[21,22] Using these metrics, the signal of the baseline vehicle was estimated to be 100.8 PLdB / 85.6 DBA while the signal of the optimized vehicle is only 92.8 PLdB / 76.7 dBA. This is a difference of over 8 dB in both metrics which is well over a factor of 2 reduction in the overall level of noise produced by the signal. In the plot of ground signal, it's apparent that not only has the height of the initial rise been reduced by more than 50%, but also the signal has been both rounded off and lengthened. Thus the profile is more sinusoidal and markedly less "N-wave" like, and the lengthened signal distributes the energy over a longer period of time. At $\alpha = 1°$, the lift coefficients on the initial and final design were 0.094 and 0.092 respectively.

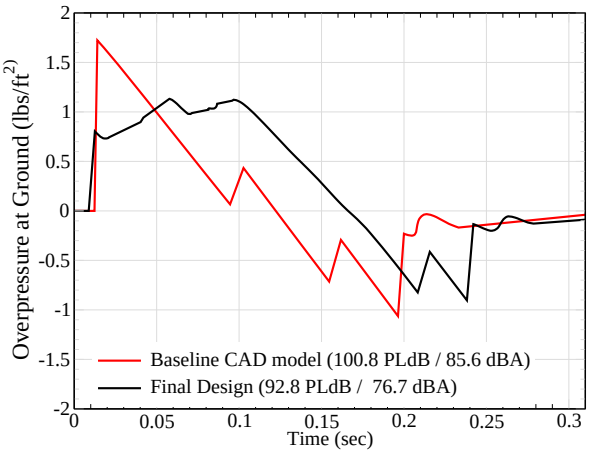

Figure 16. Ground signatures of the baseline and optimized designs for near-field signatures in Fig. 15. Propagation to the ground was performed using NFBoom.[21,22]

Considering the relative naiveté in our choice of functional, these results are quite interesting. Comparing the near-field signals in Fig. 15 to those of the F-5 and optimized Shaped Sonic Boom Demonstrator (SSBD) in ref.[26,27] there are marked similarities. In both cases, the initial rise of the baseline near-field signature has been increased mildly and is then followed by a broad, relatively flat plateau. When propagated to the ground[28] this profile gives a substantial reduction in the initial overpressure similar to what is shown at the right of Fig. 16, yielding the lower noise levels as noted above.

Figure 17 contains various views of the before and after optimization. Frame (a) contains an orthographic side-view projection of the geometry. Most dramatically, optimization has both increased the area-ruling of the fuselage and has drooped the nose to relieve the initial compression. Closer inspection reveals that it has also tailored the tail-cone of the fuselage and trimmed the horizontal tail. Frame (b) shows a perspective view looking down the nose of the vehicle which gives a better view of the shape changes to the forward fuselage and cabin. Frame (c) shows a perspective view of the underside of the vehicle. We note substantial re-contouring of the aft fuselage both ahead and aft of the wing trailing-edge. In addition, most of the bulk of the fuselage "belly" has been lifted to remove its contribution to the signature. Once again, we note a topology change at the aft wing-body juncture. In the baseline geometry the wing trailing-edge was unbroken by the fuselage. The optimized geometry has smoothed the lower aft fuselage so that it now extends beyond the root wing trailing-edge. When compared with the previous example, we see that the airfoil remains thicker since the design is being forced to carry more lift and there were no design variables open on the upper wing surface.

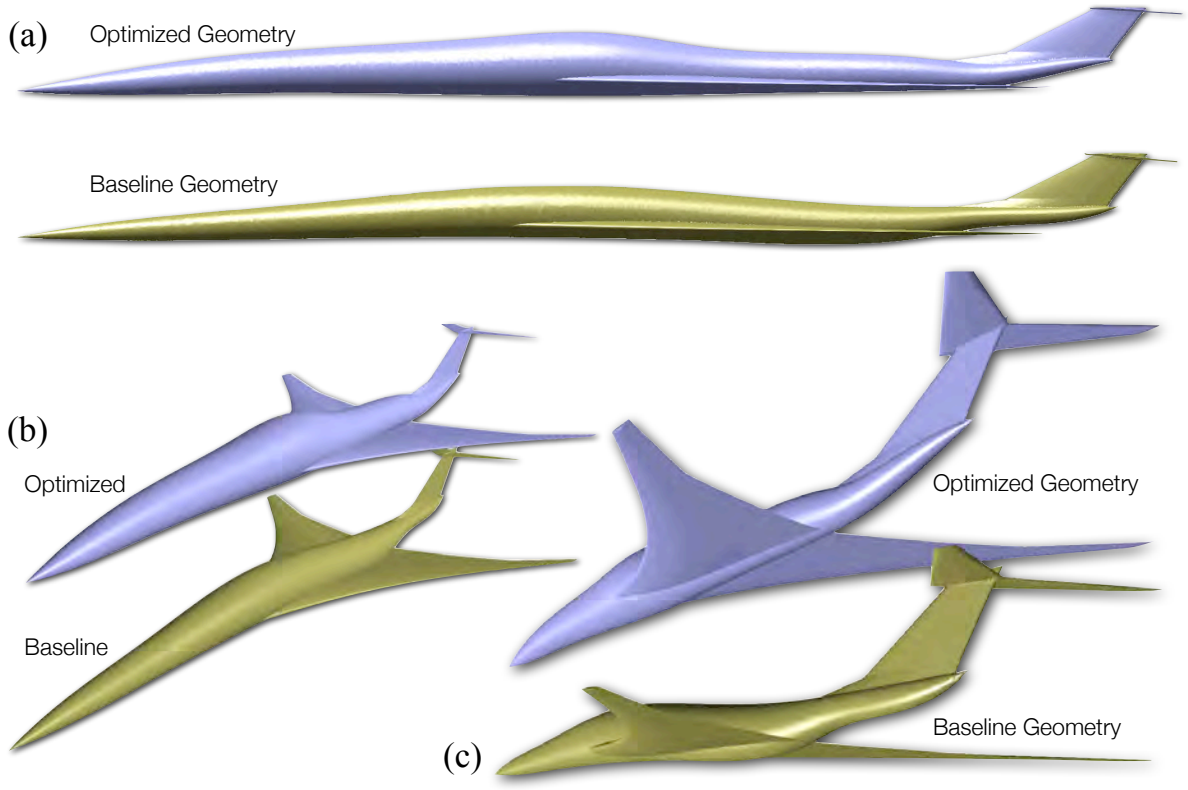

Figure 17. Initial and optimized shapes for signal driven shape optimization at fixed lift with $M_\infty = 2$ and $\alpha = 1°$.

Figure 18 shows isobars in the discrete solution of both the baseline and optimized shapes. The color map on pressure coefficient is chosen to emphasize deviation from ambient with red indicating overpressure and blue indicating underpressure. White indicates undisturbed flow. The sensors driving the adaptation are indicated on the plots and are shown 2 body-lengths below the vehicle. In comparing the pressure signatures between the initial and optimized geometries, the disturbances under the entire optimized vehicle are clearly much weaker. In particular, the strong underwing compression of the initial configuration has been entirely eliminated, and the aft expansions and compressions have been substantially tailored to produce smaller disturbances.

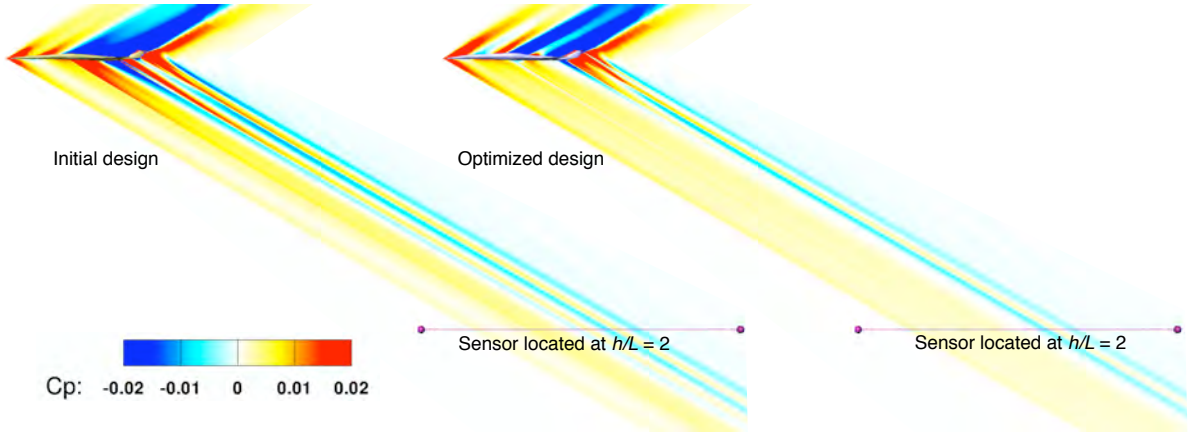

Figure 18. Contours of pressure coefficient for the initial and optimized designs for optimization at fixed lift with $M_\infty = 2$ and $\alpha = 1°$. White indicates free stream ($C_p = 0$).

C. Nacelle Integration for Transport Aircraft

The final design example is a shape optimization of a representative transport aircraft at transonic conditions. We examine the performance of the framework on a problem with a large number of design variables and relatively fine meshes. The problem involves minimizing drag at fixed lift for a configuration of four components: wing, fuselage, pylon and flow-through nacelles[f]. There are two design points, which are relatively close in Mach number but with different target lift coefficients. We also constrain the rolling moment at both design points such that, on a semispan configuration, the ratio of the rolling-moment and lift coefficients C_ℓ/C_L is about 0.38, *i.e.* the center of lift is near a parabolic lift distribution. This mimics a root-bending-moment criterion that would be imposed on a realistic transport design:

Design Point A: $M = 0.785$, $C_L = 0.52$, and $C_\ell = 0.198$

Design Point B: $M = 0.790$, $C_L = 0.485$ and $C_\ell = 0.184$

The primary design point is A, therefore the discussion and figures in this section are primarily focused on this flow condition. Similar results are obtained at design point B.

The baseline configuration is shown in Fig. 19. The configuration is constructed by the use of two in-house geometry modelers; one builds the wing and the other the nacelle-pylon assembly. The fuselage is a static component with a fixed triangulation. This is a good example of the framework's ability to combine components from an array of different modelers within the same analysis. While the wing modeler was designed to generate surface shape sensitivities automatically, the nacelle-pylon modeler is a venerable Fortran 77 code that was originally designed for use with panel methods. The framework automatically tests for the presence of surface shape sensitivities. If this test fails, then "black-box" finite-difference approximations are automatically used[g].

To establish the baseline geometry, a wing-body optimization was first performed in isolation. The nacelle-pylon assembly was added to the optimized wing at small toe-in and incidence angles. There are a total of 161 design variables. The wing shape is controlled by seven airfoil sections at stations shown in Fig. 20. The shape of each airfoil is controled by a Kulfan[29] parameterization with 20 shape parameters. We also allow variation of the twist angle at six of the stations (root twist is held fixed). Geometric details of the nacelle-pylon assembly are shown in the left inset of Fig. 20. This assembly is controlled by a total of 13 design variables, namely, nacelle incidence, nacelle and pylon camber, toe-in angle, and nine Kulfan parameters that control the shape of the outer-mold-line of the nacelle. The baseline shape of the nacelle's cross-section is shown at the bottom (transparent) view in Fig. 20. Lastly, the angle of attack at each design point is also a design variable.

[f]Propulsion effects will be added in the near future

[g]Presently, the "black-box" finite-differences are limited to surface triangulations with the same topology and connectivity at the perturbed and baseline states

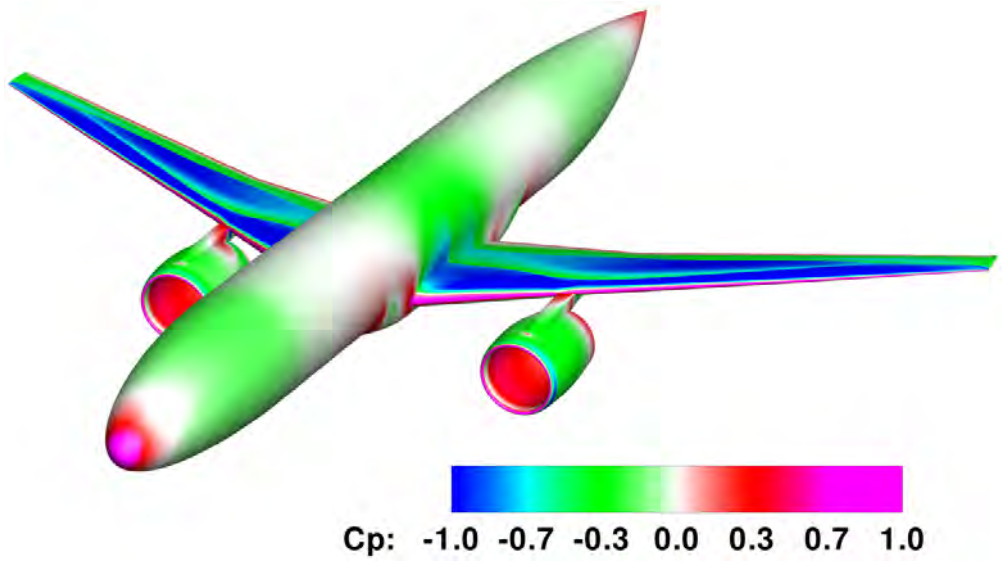

Figure 19. Baseline geometry (C_p contours, design point A: $M = 0.785$, $C_L = 0.52$)

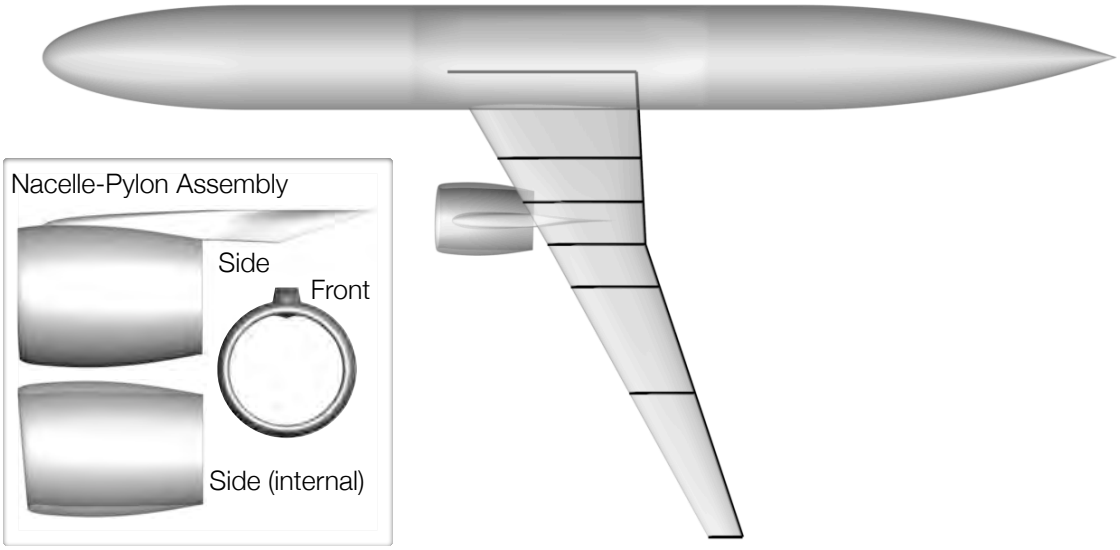

Figure 20. Top-view of baseline with wing shape-control stations indicated. Left inset shows details of nacelle geometry. There are a total of 159 geometric design variables

The objective function is a sum of the following terms at design point A:

$$\mathcal{J}_D = 200(C_D - 0.005) \tag{19}$$
$$\mathcal{J}_L = 650(C_L - 0.520)^2 \quad \text{if } C_L < 0.52 \tag{20}$$
$$\mathcal{J}_R = 650(C_\ell - 0.198)^2 \quad \text{if } C_\ell > 0.198 \tag{21}$$

and at design point B:

$$\mathcal{J}_D = 200(C_D - 0.005) \tag{22}$$
$$\mathcal{J}_L = 650(C_L - 0.485)^2 \quad \text{if } C_L < 0.485 \tag{23}$$
$$\mathcal{J}_R = 650(C_\ell - 0.184)^2 \quad \text{if } C_\ell > 0.184 \tag{24}$$

The lift and rolling moment functionals are one-sided, e.g. active only when lift is below its target value. We also impose several geometric constraints. Both nacelle diameter and pylon thickness are fixed. In addition, the maximum thickness of each airfoil control section is constrained to be equal to the baseline geometry. The geometric constraints are added to the aerodynamic objectives via quadratic penalty terms to form a single scalar objective. The objective is minimized with a quasi-Newton (BFGS) method with a backtracking line-search.

Our strategy for solving this optimization problem involves a two-level "grid-sequencing" approach. First, we use a mesh with about 8.8 million cells (semispan configuration) and solve the optimization to convergence. We then restart from the best design on a finer mesh with about 17.3 million cells. Running exclusively on the fine mesh was needlessly expensive, nevertheless this mesh was quick to eliminate a few lingering shocks that remained in the best coarse mesh design. For the results reported below, the optimization executed 57 design iterations on the coarse mesh and 12 design iterations on the fine mesh. The objective function appears to be converged and the gradient norm has been reduced by about two orders of magnitude.

As shown in Fig. 19, the baseline geometry generates a strong, double shock system on the upper surface of the wing, and there is also significant flow interference from the nacelle-pylon assembly. Examination of the flow on the lower surface reveals another shock system in the trench[h] at the pylon-wing junction, as shown in Fig 21. We also observe strong shocks at the lips of the nacelles.

The aerodynamic performance of the optimized geometry is summarized in Tables 1 and 2. Significant drag reduction is achieved at both design points: drag is reduced by 17 and 15 counts at design points A and B, respectively. The center of lift drifted just slightly outboard, most likely due to induced drag benefits. All geometric constraints are essentially satisfied, with the largest violation occuring at the wing root section where the maximum thickness is 13.96% instead of the desired 14%.

Table 1. Aerodynamic performance of initial and final designs at design point A.

Geometry	C_L	C_D	C_ℓ	C_ℓ/C_L
Initial	0.52	0.0160	0.207	0.398
Optimized	0.52	0.0143	0.209	0.402

Table 2. Aerodynamic performance of initial and final designs at design point B.

Geometry	C_L	C_D	C_ℓ	C_ℓ/C_L
Initial	0.485	0.0150	0.192	0.396
Optimized	0.485	0.0135	0.194	0.400

Figure 22 compares the initial and final pressure distributions over the top surface of the wing. The double shock system has merged into a weaker single shock. This is confirmed in Fig. 23, which shows the section pressure distributions at two locations, namely, an inboard section close to the wing-body junction and an outboard section roughly halfway between the pylon and the wing-tip. At both stations, the optimized design eliminates the double shock and shows a weaker single shock.

[h]We refer to the region where a channel flow exists between the nacelle and the lower wing surface as the "trench".

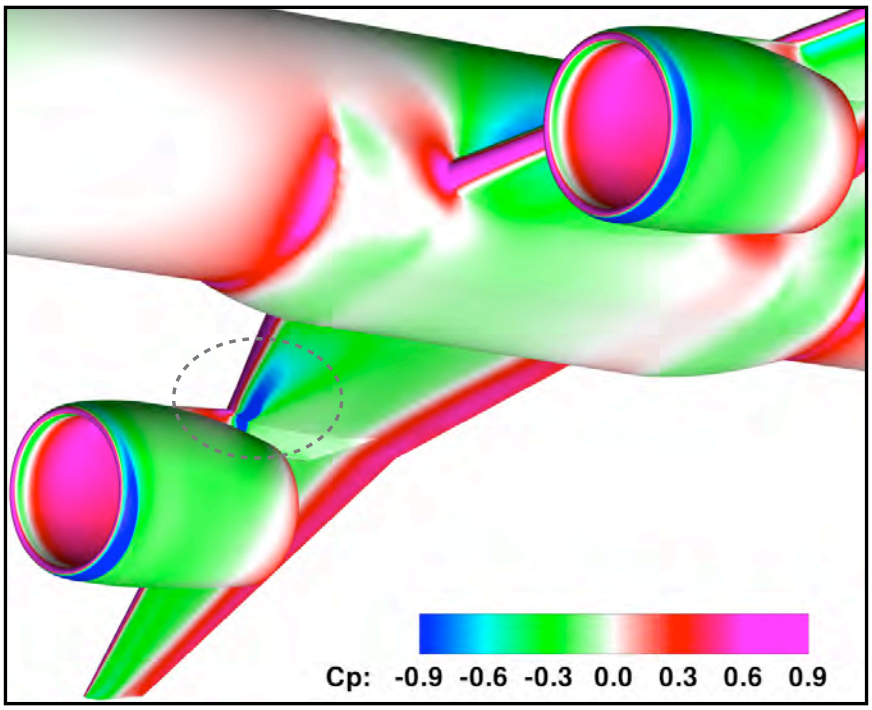

Figure 21. Close-up view of lower-surface near body region of the baseline geometry (Design point A: $M = 0.785$, $C_L = 0.52$). Strong shocks are observed in the trench (pylon-wing junction, circled) and the lip regions of the nacelles.

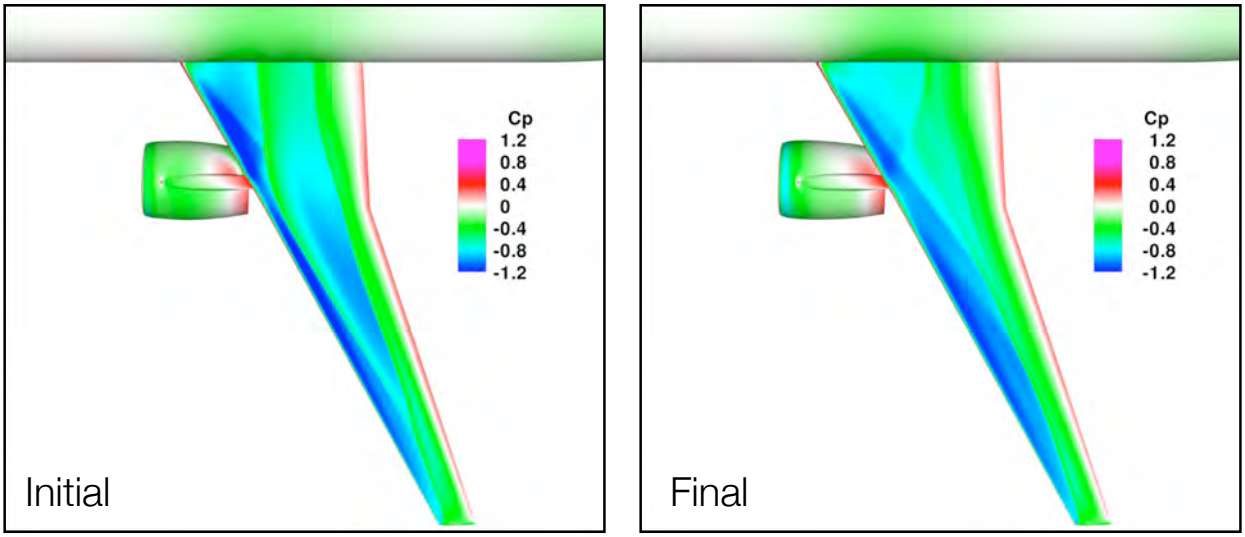

Figure 22. Initial and final C_p contours, wing top-view. The shock system is significantly weakened and the double shock has almost merged into a single shock (Design point A: $M = 0.785$, $C_L = 0.52$).

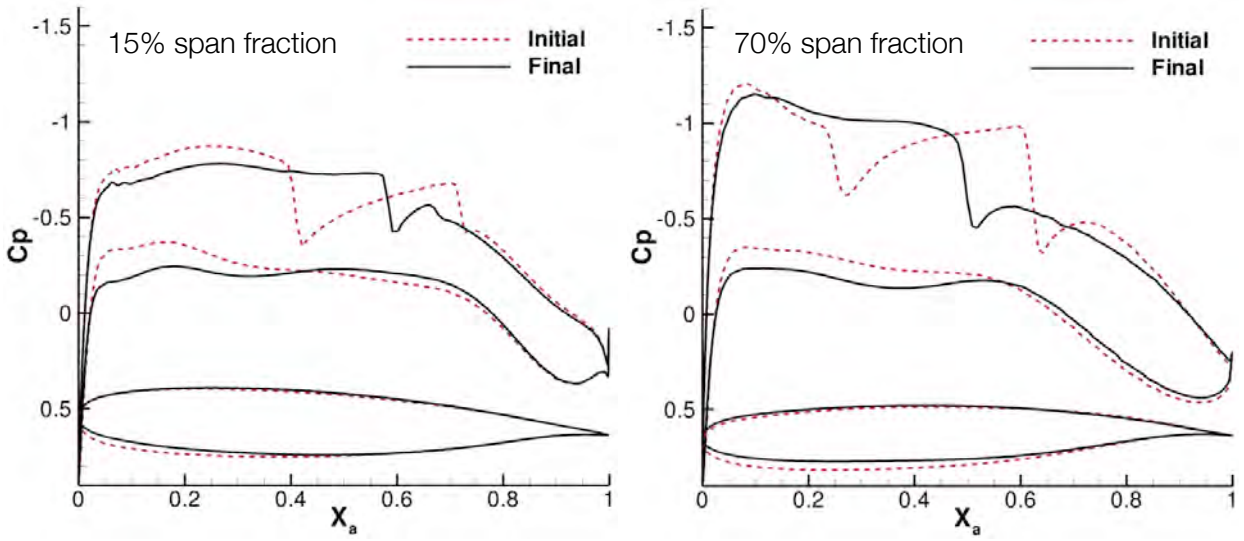

Figure 23. C_p **cuts and airfoil shapes close to the wing-body junction (left, span fraction is from fuselage centerline) and roughly halfway between the pylon and wing-tip (right). Double shocks have been eliminated and the shock strength is reduced (Design point A:** $M = 0.785$, $C_L = 0.52$**).**

Perhaps the most interesting region is the trench at the pylon-wing junction. The lower surface view in Fig. 24, and the pressure distribution on a wing section near the junction in Fig. 25, show that the interference shock has been eliminated. This is accomplished by flattening of the outer-mold-line of the nacelle, reducing its incidence, as well as thinning of the wing's leading edge. The pressure distribution of the final design is actually shock-free at this section. The optimization appears to be shifting the lift from the nacelle to the wing. This leads to interesting questions regarding optimal spanwise loadings that we leave for future work.

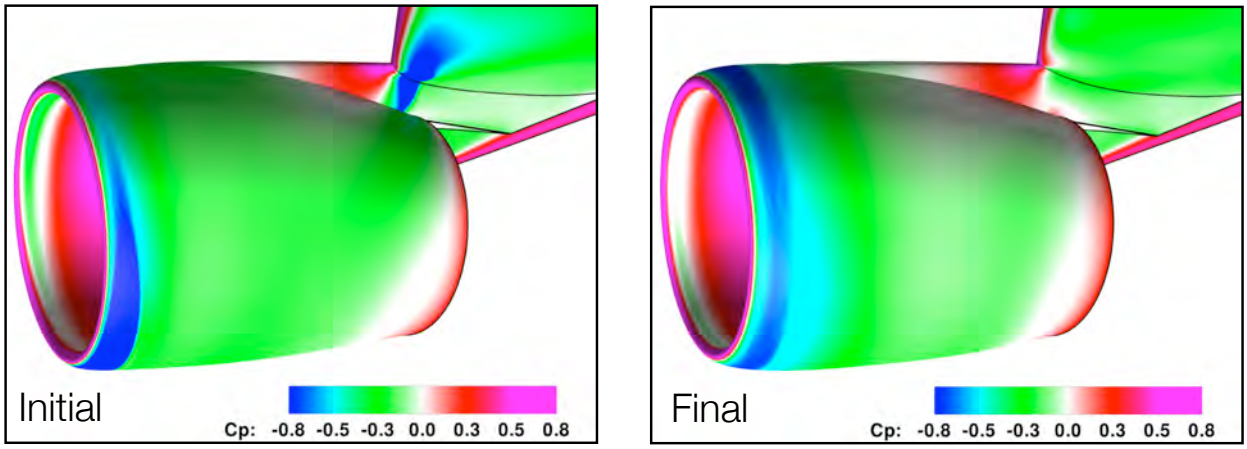

Figure 24. Close-up view of the trench region for the initial and final geometry. The shock in the trench has been eliminated and the shock on the nacelle has been weakened (Design point A: $M = 0.785$, $C_L = 0.52$**).**

We briefly summarize the performance of the framework for this case. The optimization problem involved 161 design variables and two design points and was run on 128 dual-core Intel Itanium2 processors. Furthermore, in this case, the geometry modelers share the computational resources with the flow and adjoint solvers, *i.e.* the framework is running in synchronous mode where all geometry processing completes before the flow solution begins. Table 3 gives a coarse-grained breakdown of wall-clock time for design cycles on each of the two meshes used in this problem. The first column reports the geometry generation and linearization time. It is identical for both meshes since both used the same triangulation setup yielding a surface with 501k triangles. The column labeled "Flow & Adjoint" gives the combined time of the flow and

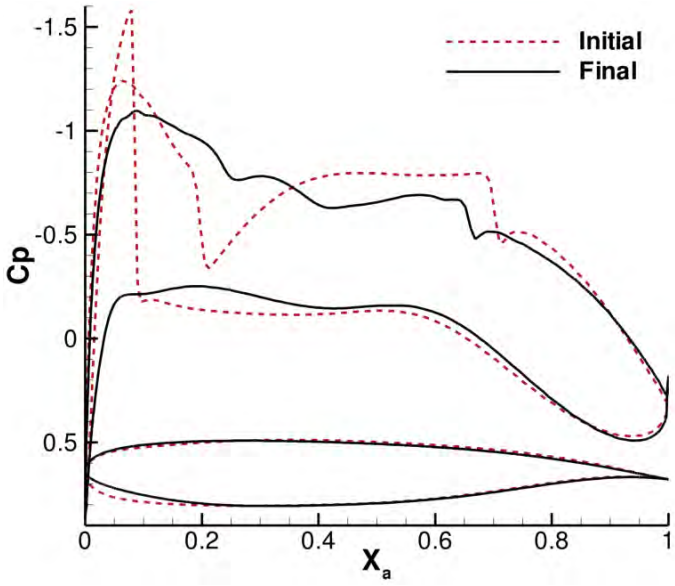

Figure 25. C_p **cuts and airfoil shapes just inboard of the pylon-wing junction (34% span). Shocks on both upper and lower surfaces have been eliminated, in particular the strong shock in the trench (Design point A:** $M = 0.785$**,** $C_L = 0.52$**).**

adjoint solvers for the two design points. The analysis time has roughly doubled consistent with doubling of the mesh. The column "Gradient" reports the time required to form the partial derivatives, including the volume mesh sensitivities, as well as the final inner product with the adjoint variables. This term is evaluated at each design point for each design variable. We observe a reduction in scalability on the fine mesh in the gradient computation. Nevertheless, the design-cycle time in these calculations is encouraging, considering the large number of design variables and relatively fine meshes.

Table 3. Wall-clock time per design cycle in minutes for the nacelle integration problem (two design points, 161 design variables, 128 dual-core Intel Itanium2 processors).

Mesh Size $\times 10^6$ cells	Geometry & Intersect	Flow & Adjoint	Gradient	Total
8.8	3.2	34.2	38	75.4
17.3	3.2	64	97	164.2

VI. Conclusions and Future Work

We have presented a parallel adjoint framework for aerodynamic shape optimization problems. The framework uses a novel adjoint formulation for an embedded-boundary Cartesian mesh method. We extended this formulation to component-based geometry, which gives designers flexibility in their choice of geometry modeler and parameterization. The first two test cases used a parametric CAD model, while the last test case used two different in-house modelers for different components of the airplane configuration. The implementation of the framework relies on a multilevel parallel strategy for performance and a new design description markup for data flow control. We demonstrated that multilevel parallelism is an efficient and practical way to achieve fast design cycle turn-around for realistic optimization problems. Future work will focus on incorporating an adjoint-based error-estimation and mesh refinement capability within the shape optimization procedure.

VII. Acknowledgments

The authors gratefully acknowledge Steve Smith (NASA Ames) for providing the nacelle integration test case, his careful testing of the framework and his support of this work. The authors would also like to thank Mathias Wintzer (Stanford University) for helpful discussions and the use of his wing geometry modeler on the transport test case; Veronica Hawke (STC) for creating the CAD model of the supersonic jet, and Kyle Washabaugh (Stanford University) and George Anderson (Stanford University) for their help with finite-difference approximations of surface shape sensitivities. This work was supported by the NASA Ames Research Center contract NNA10DF26C, and the Subsonic Fixed Wing and Supersonics projects of NASA's Fundamental Aeronautics program.

References

[1]Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, 1988, pp. 233–260, Also ICASE report 88–64.

[2]Giles, M. B. and Pierce, N. A., "An Introduction to the Adjoint Approach to Design," *Flow, Turbulence and Combustion*, Vol. 65, No. 3/4, 2000, pp. 393–415.

[3]Nemec, M. and Zingg, D. W., "Newton–Krylov Algorithm for Aerodynamic Design Using the Navier–Stokes Equations," *AIAA Journal*, Vol. 40, No. 6, 2002, pp. 1146–1154.

[4]Peter, J. E. V. and Dwight, R. P., "Numerical Sensitivity Analysis for Aerodynamic Optimization: A Survey of Approaches," *Computers & Fluids*, Vol. 39, 2010, pp. 373–391.

[5]Nemec, M. and Aftosmis, M. J., "Adjoint Sensitivity Computations for an Embedded-Boundary Cartesian Mesh Method," *Journal of Computational Physics*, Vol. 227, 2008, pp. 2724–2742.

[6]Aftosmis, M. J., Berger, M. J., and Melton, J. E., "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry," *AIAA Journal*, Vol. 36, No. 6, 1998, pp. 952–960.

[7]Nemec, M., Aftosmis, M. J., and Pulliam, T. H., "CAD-Based Aerodynamic Design of Complex Configurations Using a Cartesian Method," AIAA Paper 2004–0113, Reno, NV, Jan. 2004.

[8]Aftosmis, M. J., Berger, M. J., and Adomavicius, G., "A Parallel Multilevel Method for Adaptively Refined Cartesian Grids with Embedded Boundaries," AIAA Paper 2000–0808, Reno, NV, Jan. 2000.

[9]Aftosmis, M. J., Berger, M. J., and Murman, S. M., "Applications of Space-Filling-Curves to Cartesian Methods for CFD," AIAA Paper 2004–1232, Reno, NV, Jan. 2004.

[10]Berger, M. J., Aftosmis, M. J., , Marshall, D. D., and Murman, S. M., "Performance of a New CFD Flow Solver Using a Hybrid Programming Paradigm," *J. of Parallel and Distributed Computing*, Vol. 65, 2005, pp. 414–423.

[11]Nemec, M., Aftosmis, M. J., Murman, S. M., and Pulliam, T. H., "Adjoint Formulation for an Embedded-Boundary Cartesian Method," AIAA Paper 2005–0877, Reno, NV, Jan. 2005.

[12]Nemec, M. and Aftosmis, M. J., "Adjoint Algorithm for CAD-Based Shape Optimization Using a Cartesian Method," AIAA Paper 2005–4987, Toronto, ON, June 2005.

[13]Aftosmis, M. J., "Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries," Lecture notes, von Karman Institute for Fluid Dynamics, Series: 1997-02, Brussels, Belgium, March 1997.

[14]Eldred, M. S., Hart, W. E., Schimel, B. D., and van Bloemen Waanders, B. G., "Multilevel Parallelism for Optimization on MP Computers: Theory and Experiment," AIAA Paper 2000–4818, Albuquerque, NM, 2000.

[15]Parashar, S., English, K., and Bloebaum, C. L., "Data Transmission in Multidisciplinary Design Optimization Using a Platform-Independent Data Structure," AIAA Paper 2004–450, Reno, NV, Jan. 2004.

[16]Gloukhov, R., Tribes, C., Trépanier, J. Y., and Ozell, B., "Towards Web Standards-Based MDA and MDO," AIAA Paper 2007–966, Reno, NV, Jan. 2007.

[17]Nemec, M. and Aftosmis, M. J., "Adjoint Error Estimation and Adaptive Refinement for Embedded-Boundary Cartesian Meshes," AIAA Paper 2007–4187, Miami, FL, June 2007.

[18]Haimes, R. and Crawford, C., "Unified Geometry Access for Analysis and Design," Tech. rep., 12th International Meshing Roundtable, Santa Fe, NM, Sept. 2003.

[19]Haimes, R. and Aftosmis, M. J., "On Generating High Quality "Water-tight" Triangulations Directly from CAD," Tech. rep., Meeting of the International Society for Grid Generation, (ISGG), Honolulu, HI, June 2002.

[20]Gill, P. E., Murray, W., Michael, and Saunders, A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, Vol. 12, 1997, pp. 979–1006.

[21]Stevens, S. S., "Perceived Level of Noise by Mark VII and Decibels (E)," *The Journal of the Acoustical Society of America*, Vol. 51, No. 2B, February 1972, pp. 575–601.

[22]Durston, D. A., "A Preliminary Evaluation of Sonic Boom Extrapolation and Loudness Calculation Methods," *High-Speed Research: Sonic Boom*, NASA CP 10133, NASA Ames Research Center, May 12-14 1993, pp. 301–323.

[23]Thomas, C. L., "Extrapolation of Wind-Tunnel Sonic Boom Signatures without Use of a Whitham F-function," , No. SP-255, 1970, pp. 205–217.

[24]Thomas, C., "Extrapolation of Sonic Boom Pressure Signatures by the Waveform Parameter Method," NASA TN-D-6832, June 1972.

[25]Plotkin, K. J., "Review of Sonic Boom Theory," *AIAA 12th Aeroacoustics Conference*, No. 89-1105, AIAA, San Antonio, TX, April 1989.

[26]Wintzer, M., Nemec, M., and Aftosmis, M. J., "Adjoint-Based Adaptive Mesh Renement for Sonic Boom Prediction," *26th AIAA Applied Aerodynamics Conference*, No. 2008-6593, AIAA, Honolulu, HI, August 2008.

[27]Meredith, K. B., Dahlin, J. H., Graham, D. H., Malone, M., Haering, Jr., E. A., Page, J. A., and Plotkin, K. J., "Computational Fluid Dynamics Comparison and Flight Test Measurement of F-5E Off-Body Pressures," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, No. 2005-0006, AIAA, Reno, NV, January 2005.

[28]Maute, K., Farhat, C., Argrow, B., and Nikbay, M., "Sonic Boom Mitigation via Shape Optimization Using an Adjoint Method and Application to a Supersonic Fighter Aircraft," *European Journal of Computational Mechanics*, Vol. 17, 2008, pp. 217–243.

[29]Kulfan, B., "Universal Parametric Geometry Representation Method," *Journal of Aircraft*, Vol. 45, No. 1, Jan.-Feb. 2008, pp. 142–158.

Appendix

A. Extensible-Design-Description-Markup (XDDM)

Selected XDDM *elements* and *attributes* are presented below to give an example of data flow in the optimization framework. The syntax is compatible with most XML parsers. We prefer XPath and use implementations in libxml2, Perl (XML::LibXML module) and Java.

Variable

The element *Variable* denotes a parameter that is a design variable. This can be any driving parameter of the modules used in the optimization, e.g. a shape control parameter or angle of attack.

<Variable ID="%s" Value="%g" [Min="%g"] [Max="%g"] [TypicalSize="%g"] [Comment="%s"]/>

where

ID	Unique ID (string); *required*
Value	Design variable value (double); *required*
Min	Minimum allowable parameter value (double); *optional*
Max	Maximum allowable parameter value (double); *optional*
TypicalSize	Order-of-magnitude size, primarily used for scaling (double); *optional*
Comment	Brief description of parameter; *optional*

Analysis

The element *Analysis* denotes a driven parameter of a given application, e.g. airfoil thickness output by a geometry modeler or aerodynamic coefficients output by a CFD solver. This element usually depends on the values of the design variables, as well as other parameters.

<Analysis ID="%s" Value="%g" [Sensitivity="None | Required"] [Comment="%s"]>
[<SensitivityArray> ... </SensitivityArray>]
</Analysis>

where

ID	Unique ID (string); *required*
Sensitivity	Request linearization of this element with respect to all design variables. If omitted then "None" is assumed; *optional*
Comment	Brief description of parameter; *optional*
Value	Analysis parameter value (double); *required* Since each *Analysis* element is associated with an application, this attribute is filled-in whenever the application executes.
SensitivityArray	Child element that holds the derivative values of the analysis parameter with respect to all design variables (array of doubles). Each entry has the following syntax: <Sensitivity ID="%s" Value="%g"/> where ID is the design variable ID with respect to which the derivative is taken; *required*

Function

The element *Function* allows users to define objective functions, constraints and other custom outputs. The user may use any of the *Constant*, *Variable* and *Analysis* elements as arguments. The function is specified as a symbolic expression. If sensitivities are required then the expression is symbolically differentiated.

<Function ID="%s" Type="%s" Value="%g" [Sensitivity="None | Required"] [Comment="%s"]>
<Sum P="%s,%s,..." [T="%g,%g,..."] [W="%g,%g,..."] Expr="%s" [Min="%g,%g"] [Max="%g,%g"]/>
[<SensitivityArray> ... <SensitivityArray/>]
</Function>

where

ID	Unique ID (string); *required*
Type	Function type: present support allows "Sum" type only; *required*
Sensitivity	Request linearization of this element with respect to all design variables. If omitted then "None" is assumed; *optional*
Comment	Brief description; *optional*
Value	Function value (double); *required*
Sum	This element defines a composite scalar function, where an algebraic expression is applied over a set of Constant, Variable or Analysis parameters. This element is essentially a *foreach* loop over the selected parameters. We find that in many instances we need to apply the same algebraic expression to many Analysis parameters, e.g. when assembling penalty terms.

Sum Attribute Definitions

P	Comma delimited list of IDs corresponding to Constant, Variable or Analysis parameters
T	Comma delimited list of "targets" (user specified constants, doubles)
W	Comma delimited list of "weights" (user specified constants, doubles)
Min	Comma delimited list of minimum allowable values for each parameter in **P** list; *optional*
Max	Comma delimited list of maximum allowable values for each parameter in **P** list; *optional*
Expr	Symbolic function expression. May contain standard math functions, e.g. log, exp, trig, ^ (power), and be any combination of **P**, **T**, **W** and numbers. The expression does not have to contain all three (**P**, **T** and **W**) parameters, but if any are used then the number of elements in **P**, **T**, **W**, **Min** and **Max** vectors must match; *required*

The **Value** of the *Function* is the sum of **Expr** over each set of **P**, **T** and **W** attributes. The peculiar "PTW" form, as explained above, is a restriction to limit the implicit differentiation rules when dealing with sensitivities. We plan to lift this restriction in the future. We use the **Min** and **Max** attributes to specify one-sided constraints. For example, an optimization goal may be to minimize drag of a body subject to a minimum volume requirement. Hence, any volume greater than or equal to the minimum volume is satisfactory. We create the desired function by the use of the **Min** attribute as follows: If the **Value** of the volume parameter is greater than **Min** then the **Min** value is used in the expression, i.e. P := min(P, *MIN*). Therefore, **Expr** would evaluate to zero for volumes greater than *MIN*. Similarly, the user would use **Max** for one-sided constraints from below. In this case, if the **Value** of the parameter is less than **Max** then the **Max** value is used in the expression, i.e. P := max(P, *MAX*). The purpose of these **Min** and **Max** attributes is similar to the ones defined for design variables, except that in the design variable context they define the feasible design region ("must-not-exceed-value").

Objective

The element *Objective* allows users to combine *Function* elements to define objective functions.

<Objective ID="%s" Expr="%s" Value="%g" [Comment="%s"]/>

ID	Unique ID (string); *required*
Expr	Symbolic function expression containing function IDs. The expression is limited to sums or differences of functions. The functions may be multiplied or divided by constants; *required*
Comment	Brief description; *optional*
Value	Objective value (double); *required*

The classic constrained optimization problem has a single (scalar) objective function and one or more constraints. By using different **ID**s, the user can specify multiple objective problems. A sum is implied for *Objective* elements with the same **ID**. This is useful when a single objective function is made up of different penalty terms that are incrementally defined in the various software modules being used in the optimization. The element *Constraint* can be used to specify constraints. It has the same attributes as the *Objective* element.

Example

Below is an XDDM file for the nacelle component of the transport airplane test case.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Model ID="nacelle" Modeler="nacellemaker" Wrapper="nacellemaker_wrap.csh">
  <Constant ID="1" Comment="zte" Value="-8.1" Min="-8.6" Max="-7.9" TypicalSize="0.1"/>
  <Variable ID="2" Comment="camber" Value="0.007544" Min="-0.005" Max="0.04" TypicalSize="0.01"/>
  <Variable ID="5" Comment="incidence" Value="-0.17396" Min="-2" Max="4" TypicalSize="1.0"/>
  <Variable ID="6" Comment="knot_0" Value="0.16712" Min="0.05" Max="0.5" TypicalSize="0.1"/>

  <Analysis ID="max_nacelle_diameter" Sensitivity="Required" Value="8.610924">
    <SensitivityArray>
      <Sensitivity P="2" Value="0"/>
      <Sensitivity P="5" Value="0.0087999"/>
      <Sensitivity P="6" Value="0.4828839"/>
    </SensitivityArray>
  </Analysis>

 <Function ID="Penalty" Type="Sum" Sensitivity="Required" Value="8.53776e-05">
    <Sum P="max_nacelle_diameter" T="8.61" W="100." Expr="W*(P-T)^2"/>
    <SensitivityArray>
      <Sensitivity P="2" Value="0"/>
      <Sensitivity P="5" Value="0.0016262"/>
      <Sensitivity P="6" Value="0.0892369"/>
    </SensitivityArray>
  </Function>

 <Objective ID="MaxDiameterPenalty" Expr="Penalty" Value="8.53776e-05">
    <SensitivityArray>
      <Sensitivity P="2" Value="0"/>
      <Sensitivity P="5" Value="0.0016262399999999974"/>
      <Sensitivity P="6" Value="0.0892369514594668"/>
    </SensitivityArray>
  </Objective>
</Model>
```